

**TFM del Máster Online en Ingeniería Informática**

Gestión web de bases de datos relacionales

Presentado por José Ignacio Huidobro del Arco

en Universidad de Burgos - 11 de enero de 2019

Tutora: María Belén Vaquerizo García

1. Índice general

Apéndice A 11

1 Plan de Proyecto Software 11

1.1. Introducción 11

1.2. Planificación temporal 11

1.3. Estudio de viabilidad 12

Apéndice B 15

2 Especificación de Requisitos 15

2.1. Introducción 15

2.2. Objetivos generales 15

2.3. Catálogo de requisitos 16

2.4. Especificación de requisitos 19

Apéndice C 37

3 Especificación de diseño 37

3.1. Introducción 37

3.2. Diseño de datos 37

3.3. Diseño procedimental 51

3.4. Diseño arquitectónico 63

Apéndice D 77

4 Documentación técnica de programación 77

4.1. Introducción 77

4.2. Estructura de directorios 77

4.3. Manual del programador 78

4.4. Compilación, instalación y ejecución del proyecto 98

4.5. Pruebas del sistema 105

Apéndice E 109

5 Documentación de usuario 109

5.1. Introducción 109

5.2. Requisitos de usuario 109

5.3. Instalación 109

5.4. Manual de usuario 110

Índice de figuras

[Figura 1: Pantalla de conexión con la base de datos 52](#_Toc535938324)

[Figura 2 : Obtener conexión 64](#_Toc535938325)

[Figura 3: Utilización de login 65](#_Toc535938326)

[Figura 4: Presentación inicial de pantalla 66](#_Toc535938327)

[Figura 5: menu.jsp 67](#_Toc535938328)

[Figura 6: Ejecutar un método del metadata de una base de datos 68](#_Toc535938329)

[Figura 7: getInoke() 69](#_Toc535938330)

[Figura 8: getListInfo() 70](#_Toc535938331)

[Figura 9: TablasAction. consulta() 72](#_Toc535938332)

[Figura 10: borrar() 74](#_Toc535938333)

[Figura 11: editar() 75](#_Toc535938334)

[Figura 12editarGuardar() 76](#_Toc535938335)

[Figura 13: Pantalla de login 111](#_Toc535938336)

[Figura 14: Pantalla inicial 112](#_Toc535938337)

[Figura 15: pantalla getTables sin argumentos 113](#_Toc535938338)

[Figura 16: Pantalla getTables de un patrónn de esquema concreto 114](#_Toc535938339)

[Figura 17: Pantalla de una consulta de tabla 115](#_Toc535938340)

[Figura 18: Editar registro de una tabla 116](#_Toc535938341)

[Figura 19: Insertar registro en una tabla 117](#_Toc535938342)

Índice de tablas

[Tabla 1: Requerimiento establecimiento de conexión 23](#_Toc535938343)

[Tabla 2: Requerimiento Datos de conexión 23](#_Toc535938344)

[Tabla 3: Requerimiento Duración de conexión 23](#_Toc535938345)

[Tabla 4: Requerimiento Almacenamiento de datos de conexión 24](#_Toc535938346)

[Tabla 5: Requerimiento Control de aceso 24](#_Toc535938347)

[Tabla 6: Requerimiento Información de conexión 25](#_Toc535938348)

[Tabla 7: Requerimiento Menú del usuario 25](#_Toc535938349)

[Tabla 8: Requerimiento Menú de información 25](#_Toc535938350)

[Tabla 9: Requerimiento Parámetros de ejecución de métodos 26](#_Toc535938351)

[Tabla 10: Requerimiento Parámetros en sesión 26](#_Toc535938352)

[Tabla 11: Requerimiento Presentación del resultset 27](#_Toc535938353)

[Tabla 12: Requerimiento Ordenación de tablas 27](#_Toc535938354)

[Tabla 13: Requerimiento Visualizar tablas de la base de datos 27](#_Toc535938355)

[Tabla 14: Requerimiento Visualizar tabla 28](#_Toc535938356)

[Tabla 15: Requerimiento Filtrado de reegistros en consulta 28](#_Toc535938357)

[Tabla 16: Requerimiento Eliminación de un registro 28](#_Toc535938358)

[Tabla 17: Requerimiento Modificación de un registro 29](#_Toc535938359)

[Tabla 18: Requerimiento Inserción de un registro 29](#_Toc535938360)

[Tabla 19: Definición de requisitos 30](#_Toc535938361)

[Tabla 20: Requisito Entorno de explotación 31](#_Toc535938362)

[Tabla 21: Requisito Lenguaje de programación 31](#_Toc535938363)

[Tabla 22: Requisito Pruebas en bases de datos 32](#_Toc535938364)

[Tabla 23: Requisito Sitema operativo 32](#_Toc535938365)

[Tabla 24: requisito Cliente navegador 32](#_Toc535938366)

[Tabla 25: Requisito Pruebas de navegación 33](#_Toc535938367)

[Tabla 26: Requisito Protección de acceso 33](#_Toc535938368)

[Tabla 27: Requisito Límite de response 33](#_Toc535938369)

[Tabla 28: Requisito Modificación solo con PK 34](#_Toc535938370)

[Tabla 29: Caso de uso Obtener conexión 54](#_Toc535938371)

[Tabla 30: Caso de uso Utilizar conexión 54](#_Toc535938372)

[Tabla 31: Caso de uso Información de la conexión 55](#_Toc535938373)

[Tabla 32: Caso de uso Obtener métodos del Metadata de la base de datos 56](#_Toc535938374)

[Tabla 33: Caso de uso Ejecutar un método del metadata de la base de datos 58](#_Toc535938375)

[Tabla 34: Caso de uso Consultar datos de una tabla 60](#_Toc535938376)

[Tabla 35: Caso de uso Eliminación del registro de una tabla 61](#_Toc535938377)

[Tabla 36: Caso de uso Actualización de los datos del registro de una tabla 62](#_Toc535938378)

[Tabla 37: Caso de uso Inserción de un registro de una tabla 63](#_Toc535938379)

[Tabla 38: Estructura de carpetas 78](#_Toc535938380)

[Tabla 39: ciclo de vida clean 101](#_Toc535938381)

[Tabla 40: ciclo de vida default 103](#_Toc535938382)

[Tabla 41: ciclo de vida site 103](#_Toc535938383)

[Tabla 42: Enlaces de ciclo de vida limpio 103](#_Toc535938384)

[Tabla 43: Enlaces de ciclo de vida default - packaging war 104](#_Toc535938385)

[Tabla 44: Enlaces de ciclo de vida site 104](#_Toc535938386)

[Tabla 45: Resultados pruebas JMeter 107](#_Toc535938387)

Apéndice A

# Plan de Proyecto Software

## Introducción

### Ámbito y recursos

Se comunica al alumno el ámbito y recursos con los que se debe plantear el desarrollo de software. Tras analizar los conocimientos previos del mismo y sus necesidades formativas, se llega a la conclusión de que se desarrollará una aplicación para entorno web, desarrollada en lenguaje de programación Java, con Netbeans IDE y software de servidor web Apache Tomcat. Se utilizará la plataforma Struts2 y Maven como apoyo al desarrollo.

### Costes y planificación temporal

Al tratarse de un proyecto educativo, no se ha realizado un estudio de costes previo.

En el capítulo siguiente se comentará la planificación temporal realizada

## Planificación temporal

Para establecer una planificación temporal adecuada, comenzamos indicando las tareas que debemos realizar e indicaremos entre ellas cuales son los hitos que debemos controlar para llevar a cabo la ejecución en plazo del proyecto.

Las tareas a realizar, típicas en cualquier creación de software, en orden de ejecución, son:

* Análisis y especificaciones
* Revisión de los requisitos

hito

* Diseño arquitectónico y de datos
* Revisión del diseño preliminar

hito

* Inspección del diseño

hito

* Codificación
* Inspección del código

hito

* Prueba de unidad

hito

* Prueba de integración

hito

* Prueba de validación

hito

## Estudio de viabilidad

Debido a la característica educativa del proyecto, no se ha considerado la vertiente económica del mismo ni se ha realizado estudio de viabilidad económica alguna.

### Viabilidad legal

El desarrollo del sofware se ha planteado mediante la utilización de herramientas de desarrollo de software libre, basado en software de libre distribución a nivel educativo y empresarial, por lo que la viabilidad legal del proyecto es total.

Apéndice B

# Especificación de Requisitos

## Introducción

A continuación realizaremos una descripción completa del comportamiento del sistema que se va a desarrollar. Se incluyen un conjunto de casos de uso que describe todas las iteracciones que tendrán los usuarios con el software. También se indicarán, si fuera necesario, los requisitos que imponen restricciones en el diseño o la implementación.

## Objetivos generales

El objetivo es plantear y desarrollar los principales pasos para la cons-trucción de una aplicación Web dinámica cuyo fin sea el mantenimiento de Bases de Datos remota, via internet, ofreciendo para ello, al usuario, un entorno amigable e intuitivo que le permitirá efectuar cualquier operación básica de mantenimiento sobre una base de datos relacional (inserciones, borrados, modificaciones, consultas,..). La página web constará de una ayuda interactiva en línea, a la que el usuario tendrá la oportunidad de acceder desde cualquiera de las páginas de la aplicación, y que le será útil tanto para poder resolver dudas sobre el manejo de la aplicacion, como en cuanto a la explicación de diversos conceptos teóricos que ayuden en la comprensión de sus operaciones. La aplicación ha de diseñarse para poder administrar cualquier tipo de bases de datos: SqlServer, MySql,.. e incluso Access, ac-cediendo y tratando los metadatos de la base de datos, para permitir al sistema a través de mecanismos JDBC obtener la información necesaria para la realización de las operaciones de mantenimiento que decida el usuario. En resúmen, dicho sistema constará de distintos elementos que permitirán la realización de operaciones básicas sobre cualquier base de datos relacional (inserciones, borrados, modificaciones, consultas,..) de un modo sencillo para el usuario, con la posibilidad de aportar ayuda interactiva para guiar al usuario, paso a paso, de forma dinámica en la realización de cualquiera de estas operaciones.

## Catálogo de requisitos

### Objetivos

El presente documento tiene como objetivo puntualizar los requerimientos funcionales y no funcionales del sistema de fichas bibliográficas que se va a implementar. Los requerimientos son sustraídos de las necesidades del usuario para un mejor desempeño durante sus procesos estudio o investigación.

### Requisitos funcionales

En esta sección se detalla los requisitos funcionales para la implementación del proyecto. Así mismo, se definirá su prioridad y su relevancia para su implementación en el sistema. A continuación se detalla los requerimientos del sistema por módulo.

| Nro | Funcionalidades | Prioridad | Exigible | |
| --- | --- | --- | --- | --- |
| Módulo de conexión | | | |
| RC01 | El sistema deberá establecer conexión con base de datos relacional mediante interface JDBC | 1 | E | |
| RC02 | Para establecer la conexión con la base de datos se proporcionará URL, nombre de usuario de base de datos y contraseña | 1 | E | |
| RC03 | La conexión se mantendrá abierta de manera indefinida, hasta que la sesión http del usuario sea cerrada por el servidor web o la máquina virtual java subyacente | 1 | E | |
| RC04 | Una conexión cerrada podrá ser reabierta de manera automática para lo que el sistema debe almacenar bajo el adecuado sistema de seguridad, la URL, el nombre de usuario y la clave | 1 | E | |
| RC05 | Ante cualquier petición de una página del sistema, se comprobará si el usuario dispone de una conexión correctamente establecida. Si la conexión no estuviera establecida, el sistema deberá redirigir la petición a la pantalla de login | 1 | E | |
| Módulo de Metadatos de Conexión y base de datos | | | |
| RM01 | Mostrar al usuario la colección de información simple obtenida de la conexión | 2 | E | |
| RM02 | Mostrar al usuario a modo de menú los distintos métodos proporcioados por los metadatos de la base de datos que se organizan en forma de ResultSet | 1 | E | |
| RM03 | Mostrar al usuario a modo de menú los distintos métodos, agrupados en categorías configurables, proporcionados por los metadatos de la base de datos que devuelven información en formato simple (sin parámetros de ejecución y que devuelvan un tipo de dato simple) | 1 | E | |
| RM04 | Proporcionar al usuario un sistema que permita introducir los parámetros de ejecución de los métodos complejos de la base de datos. | 1 | E | |
| RM05 | Proporcionar al usuario un sistema que permita guardar en sesión los valores de los parámetros de ejecución establecidos en una llamada | 2 | E | |
| RM06 | Presentar los datos del ResultSet obtenido en forma de tabla | 1 | E | |
| RM07 | Permitir al usuario ordenar las tablas presentadas por cualquiera de las columnas que la componen, tanto en orden ascdente como descendente, de forma local | 2 | E | |
| Modelo de tabla | | | |
| RT01 | Permitir visualizar cualquier tabla de la base de datos, de una manera práctica y sencilla | 1 | E | |
| RT02 | Permitir visualizar los registros y campos de cualquier tabla de la base de datos | 1 | E | |
| RT03 | Permitir al usuario la inserción de un criterio de filtrado de registros por cada campo de la tabla | 1 | E | |
| RT04 | Permitir al usuario la eliminación de un registro de la tabla | 1 | E | |
| RT05 | Permitir al usuario la modificación de un registro de la tabla | 1 | E | |
| RT06 | Permitir al usuario la inserción de un registro en la tabla | 1 | E | |

### Requisitos no funcionales

En esta sección se detalla los requisitos no funcionales referentes al proyecto. Ellos describen las necesidades tecnológicas, rendimiento y software necesarios para el uso de la aplicación. A continuación se describen los requerimientos no funcionales del sistema.

| Nro | Descripción | Prioridad | Exigible |
| --- | --- | --- | --- |
| RN01 | El software se desarrollará en una plataforma Web J2EE  (Java 2 Enterprise Edition) según el patrón arquitectónico  MVC (Modelo Vista Controlador). | 1 | E |
| RN02 | El lenguaje de programación de desarrollo del proyecto será Java Web. | 1 | E |
| RN03 | La aplicación deberá ser probada al menos para las bases de datos Oracle, MySQL y SQLServer | 1 | E |
| RN04 | El sistema operativo que soportará la plataforma es Windows | 1 | E |
| RN05 | El sistema deberá ser manejado mediante un navegador o browser. | 1 | E |
| RN06 | Los navegadores comprobados para el manejo del sistema son: internet explorer (IE) versión 11, mozilla versión 64 y Chrome versión 71.0 | 1 | E |
| RN07 | La información estará protegida contra los accesos a usuarios no autorizadores. | 2 | E |
| RN08 | Con el objeto de no demorar en exceso las respuestas del servidor ante alguna consultas, se establece un límite de 1000 registros en aquellas consultas a la base de datos. Si se excede dicho límite, el sistema generará una excepción controlada informando al usuario de dicha incidencia sin presentar dato alguno. | 2 | E |
| RN09 | Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primery Key | 1 | E |

## Especificación de requisitos

[1]

### Introducción

#### Propósito

El propósito del documento es recopilar todos los requerimientos de la aplicación a desarrollar y está dirigido al desarrollador para una correcta implementación de la aplicación y al lector en general para la mejor comprensión del uso y funcionamiento de la aplicación

#### Ámbito de Sistema

Se asigna el nombre de MiDatabaseJC como nombre de la aplicación. JC son los acrónimos de Java Connection. Como se ha explicado en los objetivos, la aplicación permite conectarse a cualquier aplicación de la que exista un driver JDBC y realizar sobre ella consultas generales y modfiicaciones de datos en tablas.

Los beneficios de la aplicación pueden ser la utilización generalista, sin necesidad de instalación de software propio del fabricante.

#### Definiciones

#### Referencias

Ver el documento que engloba al actual (anexos.pdf) y memoria.pdf

### Descripción General

En esta sección se describen todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto.

#### Perspectiva del Proyecto

El proyecto puede ser considerado como un producto independiente, que no forma parte de otro proyecto superior.

En el documento Memoria.pdf se ha descrito con profundidad los productos comerciales y sus requisitos, que pueden considerarse similares en cuanto al desarrollo conceptual de la aplicación.

#### Funciones del Proyecto

Las funciones del producto deberán dar cabida a los objetivos planteados y descritos tanto en este documento como en la memoria del proyecto. Podremos agrupar las funciones en las siguientes grandes categorías

* Establecimiento de conexión JDBC con base de datos relacional
* Acceso a los metadatos de conexión y base de datos
* Acceso a los datos de la base de datos
* Modificación, eliminación o inserción de los datos de la base de datos.

#### Características de los Usuarios

El proyecto se ha diseñado para la utilización por parte de cualquier usuario que tenga unos conocimientos mínimos de las bases de datos relacionales y comprenda los conceptos de conexión, base de datos y tablas.

#### Suposiciones y Dependencias

Como ya se ha indicado en los documentos anexos, la aplicación funciona correctamente sobre plataformas Java 1.8. Se ha detectado que su funcionamiento no es adecuado, y por lo tanto no funcionará, en plataformas Java 1.7

#### Requisitos Futuros

Existen una gran cantidad de cuestiones que pueden mejorarse de la aplicación y de funciones que pueden ser ampliadas. Entre las que podemos destacar:

* Links a tablas relacionados mediante foreing keys
* Sistema de ejecución de sentencias sql
* Creación de sentencias sql complejas que afecten a varias tablas relacionadas
* Filtro de registros de tablas con posibilidad de uniones mediante operadores O

### Requisitos Específicos

Esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos, y que permita planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito aquí especificado describirá los comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

Los requisitos poserán las siguientes características:

Corrección: La ERS es correcta si y sólo si todo requisito que figura aquí (y que será implementado en el sistema) refleja alguna necesidad real. La corrección de la ERS implica que el sistema implementado será el sistema deseado.

No ambiguos: Cada requisito tiene una sola interpretación. En el caso de utilizar términos que, habitualmente, poseen más de una interpretación, se definirán con precisión en el glosario.

Completos: Todos los requisitos relevantes han sido incluidos en la ERS.

Consistentes: Los requisitos no pueden ser contradictorios. Un conjunto de requisitos contradictorio no es implementable.

Clasificados: Normalmente, no todos los requisitos son igual de importantes. Los requisitos pueden clasificarse por importancia (esenciales, condicionales u opcionales) o por estabilidad (cambios que se espera que afecten al requisito).

Verificables: La ERS es verificable si y sólo si todos sus requisitos son verificables. Un requisito es verificable (testeable) si existe un proceso finito y no costoso para demostrar que el sistema cumple con el requisito. Un requisito ambiguo no es, en general, verificable.

Modificables: La ERS es modificable si y sólo si se encuentra estructurada de forma que los cambios a los requisitos pueden realizarse de forma fácil, completa y consistente.

Trazables: La ERS es trazable si se conoce el origen de cada requisito y se facilita la referencia de cada requisito a los componentes del diseño y de la implementación.

| Identificación | RC01 |
| --- | --- |
| Nombre | Establecimiento de conexión |
| Características | El sistema deberá establecer conexión con base de datos relacional mediante interface JDBC |
| Descripción | Mediante la clase DriverManager del paquete JDBC, se establecerá la conexión con la base de datos pasándole como parámetros url de conexión, usuario y contraseña |
| Requerimiento no funcional | RN01 RN02 RN03 |
| Prioridad | 1 |

Tabla 1: Requerimiento establecimiento de conexión

|  |  |
| --- | --- |
| Identificación | RC02 |
| Nombre | Datos de conexión |
| Características | Para establecer la conexión con la base de datos se proporcionará URL, nombre de usuario de base de datos y contraseña |
| Descripción | Aunque el método getConnection de la clase DriverManager dispone de varias versiones, se utilizará el que necesita url, usuario y clave como datos necesarios para realizar la conexión |
| Requerimiento no funcional | RN01 RN02 RN07 |
| Prioridad | 1 |

Tabla 2: Requerimiento Datos de conexión

|  |  |
| --- | --- |
| Identificación | RC03 |
| Nombre | Duración de la conexión |
| Características | La conexión se mantendrá abierta de manera indefinida, hasta que la sesión http del usuario sea cerrada por el servidor web o la máquina virtual java subyacente |
| Descripción | La aplicación no deberá cerrar la conexión tras la ejecución de comandos de base de datos. La aplicación deberá cerrar la conexión cuando se ejecute el método finalze de la clase en la que esta se encuentra, fruto del proceso de recolección de basura de la máquina virtual java |
| Requerimiento no funcional | RN01 RN02 RN07 |
| Prioridad | 1 |

Tabla 3: Requerimiento Duración de conexión

|  |  |
| --- | --- |
| Identificación | RC04 |
| Nombre | Almacenamiento de datos de conexión |
| Características | Una conexión cerrada podrá ser reabierta de manera automática para lo que el sistema debe almacenar bajo el adecuado sistema de seguridad, la URL, el nombre de usuario y la clave |
| Descripción | Se utilizará una clase ConnectionImpl que, además de la conexión java, mantendrá los datos de la conexión, incluyendo la contraseña del usuario. La instanciación de la clase se realiza en el momento de crearse la conexión y será mantenida para cada usuario en el espacio de memoria de sesión mantenido por el servidor web |
| Requerimiento no funcional | RN01 RN02 RN07 |
| Prioridad | 1 |

Tabla 4: Requerimiento Almacenamiento de datos de conexión

|  |  |
| --- | --- |
| Identificación | RC05 |
| Nombre | Control de acceso |
| Características | Ante cualquier petición de una página del sistema, se comprobará si el usuario dispone de una conexión correctamente establecida. Si la conexión no estuviera establecida, el sistema deberá redirigir la petición a la pantalla de login |
| Descripción | Utilizando el método validate de las clases Action de struts2 comprobaremos que existe sesión de usuario, que tiene una conexión y que es válida y no está cerrada. En caso contrario se deberá redirigir a la pantalla de login |
| Requerimiento no funcional | RN01 RN02 RN07 |
| Prioridad | 1 |

Tabla 5: Requerimiento Control de aceso

|  |  |
| --- | --- |
| Identificación | RM01 |
| Nombre | Información de conexión |
| Características | Mostrar al usuario la colección de información simple obtenida de la conexión |
| Descripción | Se utilizarán los métodos de la clase Connection y se mostrará en pantalla al usuario la información de aquellos que devuelvan información sencilla mediante métodos que no requieran parámetros |
| Requerimiento no funcional | RN05 |
| Prioridad | 2 |

Tabla 6: Requerimiento Información de conexión

|  |  |
| --- | --- |
| Identificación | RM02 |
| Nombre | Menú del usuario |
| Características | Mostrar al usuario a modo de menú los distintos métodos proporcioados por los metadatos de la base de datos que se organizan en forma de ResultSet |
| Descripción | Se crea una clase DatabaseMetaDataImpl que implementa el interface DatabaseMetaData y todos sus métodos. Un proceso que obtiene la lista de métodos de esta clase, filtra los que devuelven ResultSet y los presenta en una página web |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 7: Requerimiento Menú del usuario

|  |  |
| --- | --- |
| Identificación | RM03 |
| Nombre | Menú de información |
| Características | Mostrar al usuario a modo de menú los distintos métodos, agrupados en categorías configurables, proporcionados por los metadatos de la base de datos que devuelven información en formato simple (sin parámetros de ejecución y que devuelvan un tipo de dato simple) |
| Descripción | En la clase DatabaseMetaDataImpl se incorporan métodos que devuelven una lista de propiedades y valores |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 8: Requerimiento Menú de información

|  |  |
| --- | --- |
| Identificación | RM04 |
| Nombre | Parámetros de ejecución de métodos |
| Características | Proporcionar al usuario un sistema que permita introducir los parámetros de ejecución de los métodos complejos de la base de datos. |
| Descripción | Para llevar a cabo este requerimiento, debe utilizarse la opción getParameters de java Reflection. Este método requiere Java 1.8. Además, se utlizará el método getParameterTypes para obtener el tipo de datos al que pertenecen los parámetros |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 9: Requerimiento Parámetros de ejecución de métodos

|  |  |
| --- | --- |
| Identificación | RM05 |
| Nombre | Parámetros en sesión |
| Características | Proporcionar al usuario un sistema que permita guardar en sesión los valores de los parámetros de ejecución establecidos en una llamada |
| Descripción | Cuando el usuario envía a la aplicación los parámetros de ejecución de un método de DatabaseMetaData, el sistema, previamente a su utilización, los deberá almacenar en la sesión del usuario en el servidor web |
| Requerimiento no funcional | RN07 RN05 |
| Prioridad | 1 |

Tabla 10: Requerimiento Parámetros en sesión

|  |  |
| --- | --- |
| Identificación | RM06 |
| Nombre | Presentación del resultse |
| Características | Presentar los datos del ResultSet obtenido en forma de tabla |
| Descripción | El programa dispondrá de un sistema que permita visualizar un resultset en forma de tabla html |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 11: Requerimiento Presentación del resultset

|  |  |
| --- | --- |
| Identificación | RM07 |
| Nombre | Ordenación de tablas |
| Características | Permitir al usuario ordenar las tablas presentadas por cualquiera de las columnas que la componen, tanto en orden ascdente como descendente, de forma local |
| Descripción | El procedimiento de ordenación debe realizarse en el navegador del cliente, no se precisa una nueva request al servidor. Se utilizará la biblioteca jquery para realizar la operación. |
| Requerimiento no funcional | RN05 RN06 |
| Prioridad | 1 |

Tabla 12: Requerimiento Ordenación de tablas

|  |  |
| --- | --- |
| Identificación | RT01 |
| Nombre | Visualizar tablas de la base de datos |
| Características | Permitir visualizar cualquier tabla de la base de datos, de una manera práctica y sencilla |
| Descripción | Mediante llamada al método getTables de la clase DatabaseMetaData con los parámetros adecuados el usuario podrá elegir la tabla que consultar |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 13: Requerimiento Visualizar tablas de la base de datos

|  |  |
| --- | --- |
| Identificación | RT02 |
| Nombre | Visualizar tabla |
| Características | Permitir visualizar los registros y campos de cualquier tabla de la base de datos |
| Descripción | El usuario enviará una sentencia SELECT \* FROM tabla y se visualizará todos los datos y todos los campos de la tabla consultada |
| Requerimiento no funcional | RN05 |
| Prioridad | 1 |

Tabla 14: Requerimiento Visualizar tabla

|  |  |
| --- | --- |
| Identificación | RT03 |
| Nombre | Filtrado de registros en consulta |
| Características | Permitir al usuario la inserción de un criterio de filtrado de registros por cada campo de la tabla |
| Descripción | Se debe permitir al usuario introducir valores que permitan filtrar los registros por todos los campos que sean posible |
| Requerimiento no funcional | RN08 |
| Prioridad | 1 |

Tabla 15: Requerimiento Filtrado de reegistros en consulta

|  |  |
| --- | --- |
| Identificación | RT04 |
| Nombre | Eliminación de un registro |
| Características | Permitir al usuario la eliminación de un registro de la tabla |
| Descripción | Se debe permitir la eliminación de un registro de una tabla en la que exista una primary key |
| Requerimiento no funcional | RN09 |
| Prioridad | 1 |

Tabla 16: Requerimiento Eliminación de un registro

|  |  |
| --- | --- |
| Identificación | RT05 |
| Nombre | Modificación de un registro |
| Características | Permitir al usuario la modificación de un registro de la tabla |
| Descripción | Se debe permitir al usuario la modificación de un registro de la tabla que disponga de una primary key |
| Requerimiento no funcional | RN09 |
| Prioridad | 1 |

Tabla 17: Requerimiento Modificación de un registro

|  |  |
| --- | --- |
| Identificación | RT06 |
| Nombre | Inserción de un registro |
| Características | Permitir al usuario la inserción de un registro en la tabla |
| Descripción | Permitir al usuario introducir un registro en la tabla |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 18: Requerimiento Inserción de un registro

| Nro | Corrección | No ambiguo | Completo | Consistente | Clasificado | Verificable | Modificable |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RC01 |  |  |  |  |  |  |  |
| RC02 |  |  |  |  |  |  |  |
| RC03 |  |  |  |  |  |  |  |
| RC04 |  |  |  |  |  |  |  |
| RC05 |  |  |  |  |  |  |  |
| RM01 |  |  |  |  |  |  |  |
| RM02 |  |  |  |  |  |  |  |
| RM03 |  |  |  |  |  |  |  |
| RM04 |  |  |  |  |  |  |  |
| RM05 |  |  |  |  |  |  |  |
| RM06 |  |  |  |  |  |  |  |
| RM07 |  |  |  |  |  |  |  |
| RT01 |  |  |  |  |  |  |  |
| RT02 |  |  |  |  |  |  |  |
| RT03 |  |  |  |  |  |  |  |
| RT04 |  |  |  |  |  |  |  |
| RT05 |  |  |  |  |  |  |  |
| RT06 |  |  |  |  |  |  |  |

Tabla 19: Definición de requisitos

### Requisitos no funcionales

Esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos, y que permita planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito aquí especificado describirá los comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

|  |  |
| --- | --- |
| Identificación | RN01 |
| Nombre | Entorno de explotación |
| Características | El software se desarrollará en una plataforma Web J2EE  (Java 2 Enterprise Edition) según el patrón arquitectónico  MVC (Modelo Vista Controlador). |
| Descripción | En concreto, el desarrollo se realizará en un servidor Apache Tomcat con una plataforma struts2 |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 20: Requisito Entorno de explotación

|  |  |
| --- | --- |
| Identificación | RN02 |
| Nombre | Lenguaje de programación |
| Características | El lenguaje de programación de desarrollo del proyecto será Java Web. |
| Descripción | El lenguaje de programación de desarrollo del proyecto será Java Web. Versión 1.8 |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 21: Requisito Lenguaje de programación

|  |  |
| --- | --- |
| Identificación | RN03 |
| Nombre | Pruebas en bases de datos |
| Características | La aplicación deberá ser probada al menos para las bases de datos Oracle, MySQL y SQLServer |
| Descripción | Versiones 11 para Oracle, SQLSERVER 2012 express y MySQL ultima versión deben ser realizadas |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 22: Requisito Pruebas en bases de datos

|  |  |
| --- | --- |
| Identificación | RN04 |
| Nombre | Sistema operativo |
| Características | El sistema operativo que soportará la plataforma es Windows |
| Descripción | El sistema operativo que soportará la plataforma es Windows |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 23: Requisito Sitema operativo

|  |  |
| --- | --- |
| Identificación | RN05 |
| Nombre | Cliente navegador |
| Características | El sistema deberá ser manejado mediante un navegador o browser. |
| Descripción | El sistema deberá ser manejado mediante un navegador o browser. |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 24: requisito Cliente navegador

|  |  |
| --- | --- |
| Identificación | RN06 |
| Nombre | Pruebas de navegación |
| Características | Los navegadores comprobados para el manejo del sistema son: internet explorer (IE) versión 11, mozilla versión 64 y Chrome versión 71.0 |
| Descripción | Los navegadores comprobados para el manejo del sistema son: internet explorer (IE) versión 11, mozilla versión 64 y Chrome versión 71.0 |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 25: Requisito Pruebas de navegación

|  |  |
| --- | --- |
| Identificación | RN07 |
| Nombre | Protección de acceso |
| Características | La información estará protegida contra los accesos a usuarios no autorizadores. |
| Descripción | Toda petición realziada deberá ser redirigida a protocolo seguro HTTPS. Tras cada petición, la aplicación debe comprobar que el usuario esté correctamente autorizado |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 26: Requisito Protección de acceso

|  |  |
| --- | --- |
| Identificación | RN08 |
| Nombre | Límite de response |
| Características | Con el objeto de no demorar en exceso las respuestas del servidor ante alguna consultas, se establece un límite de 1000 registros en aquellas consultas a la base de datos. Si se excede dicho límite, el sistema generará una excepción controlada informando al usuario de dicha incidencia sin presentar dato alguno. |
| Descripción |  |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 27: Requisito Límite de response

|  |  |
| --- | --- |
| Identificación | RN09 |
| Nombre | Modificación solo con PK |
| Características | Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primery Key |
| Descripción | Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primery Key |
| Requerimiento no funcional |  |
| Prioridad | 1 |

Tabla 28: Requisito Modificación solo con PK

### Interfaces Externas

Se describirán los requisitos que afecten a la interfaz de usuario, interfaz con otros sistemas (hardware y software) e interfaces de comunicaciones.

#### Funciones

Por jerarquía funcional, se desglosan en las siguientes categorías:

Funciones de conexión. Todas aquellas funionalidades relacionadas con el mantenimiento y consecución de la conexión con la base de datos

Funciones de Metadatos de Conexión y base de datos. Funcionalidades que exploran los metadatos de la conexión o de la propia base de datos e informan al usuario de las posibilidades que tiene en cuanto a la organización de la propia base de datos

Funciones de tabla. El usuario no solo puede explorar la base de datos, también puede acceder a los datos almacenados en sus tablas y modificarlos, borrarlos o insertar nuevos registros en las tablas.

#### Requisitos de Rendimiento

No se ha realizado un estudio de requisitos de rendimiento ni de capacidad requerida por el sistema para almacenar la información del usuario.

#### Atributos del Sistema

Fiabilidad. La fiabilidad de la aplicación se basa en la fiabilidad del entorno de ejecución en la que se desarrolla, en la fiabilidad de la base de datos que se consulta y en la fiabilidad de las comunicaciones entre ambos sistema.

Mantenibilidad. El sistema no requiere mantenibilidad ninguna y, manteniendose las condiciones iniciales de ejecución, se mantendrá el funcionamiento de la aplicación

Portabilidad. Al tratarse de una aplicación Java, en teoría, es directamente exportable a cualquier servidor web que esté dotado de la posibilidad de ejecutar aplicaciones web basadas en jave y en los estándares JSP, independientemente del sistema operativo subyacente.

Seguridad. El acceso por parte del usuario a la base de datos, y todo el conjunto de la aplicación, se realiza mediante un mapeo automático y obligatorio de todas las request desde el protocolo http hacia el protocolo https. Mediante URL, usuario y contraseña convenientemente codificado, se accederá a la base de datos. Tanto la conexión como la información de conexión son almacenadas en la sesión del usuario, por lo que toda comunicación entre cliente y aplicación estará protegida por el protocolo https.

Apéndice C

# Especificación de diseño

## Introducción

El análisis de los requisitos da lugar a la especificación software donde se concretan necesidades y restricciones con las que debe trabajar el software. Se intenta obtener uno o varios modelos que detallen el comportamiento deseado del sistema.

Trataremos de ofrecer una visión de alto nivel sin descender a explicar detalles concretos del mismo.

La técnica de modelado utilizado es la de aproximaciones sucesivas en la que tomamos como partida el modelo de un sistema similar y luego mediante la experiencia y el conocimiento del problema se van proponiendo modelos intermedios.

## Diseño de datos

A continuación se detallan las distintas clases creadas para manejar los datos de la aplicación, y se indican las funciones precisas para trabajar con ellos

### Paquete es.ubu.alu.mydatabasejc

#### Menu.java

Contiene los datos necesarios para manejar una opción del menú

public class Menu {

private String action;

private String metodo;

private String parametros;

#### PropiedadValor

Se utiliza para presentar al usuario información en forma propiedad/valor

public class PropiedadValor {

private String propiedad;

private Object valor;

public PropiedadValor(

String propiedad, Object valor);

#### ValoresPorDefecto

Esta clase contiene una serie de valores de configuración de la aplicación

de uso general que pueden ser modificados por el usuario en el futuro

public class ValoresPorDefecto {

public static int numMaxRecords = 1000;

public static String autor =

"José Ignacio Huidobro del Arco";

public static String email = "jhd1001@ubu.alu.es";

public static String titulo = "MyDatabaseJC";

public static String subtitulo =

"Trabajo Fin de Máster";

public static String estudio =

"Máster Universitario Online en Ingeniería Informática";

public static String descripcionTrabajo =

"Creción de componentes para el mantenimiento web de datos en sistemas de gestión de bases de datos relacionales";

}

### Paquete es.ubu.alu.mydatabasejc.annotations

#### MetaDataInfoCategorias

Define la anotación para categorizar los métodos de DatabaseMetaDataImpl

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface MetaDataInfoCategorias {

String categoria() default "Básica";

}

#### MetaDataLink

Define la anotación para marcar los métodos de DatabaseMetaDataImpl que deben proporcionar la información para establecer un link hacia otra pantalla y cuales han de ser los parámetros a enviar en la request de ese link

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface MetaDataLink {

// devuelve la acción struts que debe ejecutarse

String action();

// devuelve el namespace en el que debe ejecutarse la acción

String namespace();

// devuelve los parametros que deben incluirse en el link, en formato array String

String[] parametros();

// marca el número de la columna en la que debe situarse el link

int columnNumber();

}

### Paquete es.ubu.alu.mydatabasejc.exceptions

#### ConnectionException

/\*\*

\* Excepciones producidas en la conexión con la base

\* de datos

\* @author jhuidobro

\*/

public class ConnectionException extends Throwable {

final static Logger logger = (Logger) LogManager

.getLogger(ConnectionException.class);

ConnectionImpl connectionImpl;

public ConnectionException(

Throwable cause, String driver);

public ConnectionException();

public ConnectionException(

String msg, ConnectionImpl connectionImpl);

public ConnectionException(

Throwable cause, ConnectionImpl connectionImpl);

#### DatabaseMetaDataException

public class DatabaseMetaDataException

extends Throwable {

final static Logger logger = (Logger) LogManager

.getLogger(DatabaseMetaDataException.class);

public DatabaseMetaDataException();

public DatabaseMetaDataException(

String msg, String metodo);

#### MenuException

public class MenuException extends Throwable {

final static Logger logger = (Logger) LogManager

.getLogger(MenuException.class);

ConnectionImpl connectionImpl;

public MenuException(

Throwable cause, String action, String metodo);

public MenuException();

#### ResultSetException

public class ResultSetException extends Throwable {

final static Logger logger = (Logger) LogManager

.getLogger(ResultSetException.class);

public ResultSetException();

public ResultSetException(

String msg, ResultSet resultSet) throws SQLException;

#### SQLCommandException

public class SQLCommandException extends Throwable

private SQLCommand sqlCommand;

final static Logger logger = (Logger) LogManager

.getLogger(SQLCommandException.class);

public SQLCommandException(

SQLCommand aThis, int operacion);

public SQLCommandException(

SQLCommand aThis, SQLException ex);

public SQLCommandException(

String TABLE\_SCHEM, String TABLE\_NAME,

String localizedMessage);

#### TablasActionException

public class TablasActionException extends Throwable {

final static Logger logger = (Logger) LogManager

.getLogger(TablasActionException.class);

ConnectionImpl connectionImpl;

public TablasActionException(String message);

public TablasActionException(

String message, Throwable cause);

public TablasActionException(Throwable cause);

### Paquete es.ubu.alu.mydatabasejc.jdbc

#### ConnectionInfo

/\*\*

\* Define los métodos informativos de la conexión

\* @author jhuidobro

\*/

public class ConnectionInfo {

protected Connection connection;

public boolean isReadOnly() throws SQLException;

public String getSchema() throws SQLException;

public boolean getAutoCommit() throws SQLException;

public String getCatalog() throws SQLException;

public Properties getClientInfo()

throws SQLException;

public int getHoldability() throws SQLException;

public int getNetworkTimeout() throws SQLException;

public int getTransactionIsolation()

throws SQLException;

public ConnectionInfo(Connection connection);

}

#### ConnectionImpl

/\*\*

\* Establece y manteniene la información de una conexión

\* con la base de datos

\* @author jhuidobro

\*/

public class ConnectionImpl {

protected Connection connection;

private String url;

private String usuario;

private String password;

private String driver;

/\*\*

\* Cierra la conexión si estuviera establecida

\*/

public void close();

/\*\*

\* Instancia un objeto de la clase ConnectionImpl

\* @param url URL de conexión a la base de datos

\* @param usuario Usuario de la base de datos

\* @param password Contraseña del usuario

\* @throws ConnectionException Cualquier error

\* producido en el proceso

\* de conexión devuelve una excepción que es pasada

\* previamente por el log del sistema

\*/

public ConnectionImpl(

String url, String usuario, String password)

throws ConnectionException;

/\*\*

\* Comprueba si la conexión a la base de datos está

\* o no cerrada

\* @return true si la conexión está cerrada, false

\* en caso contrario

\*/

public boolean isClosed() {

if (connection==null) return true;

try {

return connection.isClosed();

} catch (SQLException ex) {

new ConnectionException(ex, this);

return true;

}

}

/\*\*

\* Comprueba si la conexión con la base de datos es

\* válida

\* @param timeout Ver el mismo parámetro de

\* java.sql.Connection.isValid

\* @return true si la conexión es una conexión

\* válida, false en caso contrario

\*/

public boolean isValid(int timeout);

/\*\*

\* Cuando el objeto es recolectado para su

\* destrucción, se cierra

\* la conexión con la base de datos, si es que

\* existía alguna.

\* @throws Throwable

\*/

@Override

protected void finalize() throws Throwable;

}

#### DatabaseMetaDataImpl

/\*\*

\* Implementa los métodos de DatabaseMetaData y se añaden

\* los métodos necesraios para agrupar los que devuelven

\* información en formato simple bajo anotaciones

\* MetaDataInfoCategorias

\* @author jhuidobro

\*/

public class DatabaseMetaDataImpl

implements DatabaseMetaData {

protected DatabaseMetaData metadata;

private Method[] getMetodos(String categoria);

public DatabaseMetaDataImpl(

DatabaseMetaData metadata);

public List<List> getBasicInfo();

public List<List> getConnInfo();

public List<List> getDataBaseInfo();

public List<List> getDataInfo();

public List<List> getUserInfo();

public List<List> getSQLInfo();

private List<List> getInfo(Method[] methods);

@Override

@MetaDataInfoCategorias(categoria = "Usuario")

public boolean allProceduresAreCallable()

throws SQLException;

...

@Override

public String getURL() throws SQLException;

...

@Override

@MetaDataLink(action = "consulta",

namespace = "/tablas",

parametros =

{"TABLE\_SCHEM","TABLE\_NAME","TABLE\_TYPE"},

columnNumber = 3)

public ResultSet getTables(

String catalog, String schemaPattern,

String tableNamePattern, String[] types)

throws SQLException;

#### SQLCommand

/\*\*

\* Clase que permite al usuario ejecutar comandos sql

\* contra la base de datos

\* @author jhuidobro

\*/

public class SQLCommand {

public static int OPERACION\_UPDATE = 1;

public static int OPERACION\_DELETE = 2;

public static int OPERACION\_INSERT = 4;

public static int OPERACION\_WHERE = 8;

public static int ISAUTOINCREMENT = 1;

public static int ISCASESENSITIVE = 2;

public static int ISCURRENCY = 4;

public static int ISDEFINITELYWRITABLE = 8;

public static int ISNULLABLE = 16;

public static int ISREADONLY = 32;

public static int ISSEARCHABLE = 64;

public static int ISSIGNED = 128;

public static int ISWRITABLE = 256;

public static int ISALL = -1;

public ConnectionImpl connectionImpl;

public String cadenaWhere;

public List<Object> listaParametrosWhere;

public String cadenaSet;

public List<Object> listaParametrosSet;

public String sql;

public String cadenaInsert;

public String cadenaInsert2;

public List<Object> listaParametrosInsert;

public static Object getValor(

int tipo, String valor);

public int executeUpdate(

String esquema, String tabla,

Map<String, Integer[]> mapa, String[] campos,

String[] valores) throws SQLCommandException;

private ResultSet executeQuery(

String TABLE\_SCHEM, String TABLE\_NAME,

int TYPE\_SCROLL, int CONCUR)

throws SQLCommandException;

public ResultSet executeQuery(

String TABLE\_SCHEM, String TABLE\_NAME,

Set<String> arrayParametros,

Map<String, Object> sesion)

throws SQLCommandException;

public void setSQLPartList(

String TABLE\_SCHEM, String TABLE\_NAME,

Set<String> arrayParametros,

Map<String, Object> sesion);

public Map<String, Integer> getMap(

Map<String, Integer[]> mapa, int tipo);

public Map<String, Integer[]> getMap(

String TABLE\_SCHEM, String TABLE\_NAME);

public ResultSet executeQuery(

String TABLE\_SCHEM, String TABLE\_NAME,

Map<String, Integer[]> mapa,

String[] pkArgumentos, String[] pkValores,

int TYPE\_SCROLL, int CONCUR)

throws SQLCommandException;

private void setSQLPartList(

Map<String, Integer[]> mapa,

String[] pkArgumentos, String[] pkValores,

int tipo);

/\*\*

\* Siempre será update

\*

\* @param pkArgumentos

\* @param pkValores

\* @param campos

\* @param valores

\*/

private void setSQLPartList(

Map<String, Integer[]> mapa,

String[] pkArgumentos, String[] pkValores,

String[] campos, String[] valores);

private String getEsquema(String esquema);

public int executeUpdate(

String esquema, String tabla,

Map<String, Integer[]> mapa, int operacion,

String[] pkArgumentos, String[] pkValores,

String[] campos, String[] valores)

throws SQLCommandException;

}

### Paquete es.ubu.alu.mydatabasejc.actions

Este paquete engloba las clases con la lógica de negocio de la aplicación y gestiona el flujo de ejecución del programa apoyadas en el fichero de configuración struts.xml

#### LoginAction

Clase de la que heredan las demás, al menos aquellas que deban validar una conexión a la base de datos (en nuestro trabajo, todas)

public class LoginAction extends ActionSupport {

protected String CONEXION = "conexion";

private String url;

private String usuario;

private String password;

public String ayuda(String propiedad)

/\*\*

\* Realiza la validación de los campos de formulario

\*/

@Override

public void validate()

/\*\*

\* Presenta siempre la pantalla para logarse

\* @return "success"

\*/

public String logear()

/\*\*

\* Se valida el intento de login. Si se produce un error en una

\* conexión

\* previamente establecida, se intentará realizar de nuevo la

\* conexión

\* con los datos previos mediante llamada al método login()

\* @return "error" si no se puede validar el login

\* "success" en caso de que el login se haya validado

\* correctametne

\*/

public String validarLogin();

/\*\*

\* Se valida el intento de login. Si se produce un error en una

\* conexión

\* previamente establecida, se intentará realizar de nuevo la

\* conexión

\* con los datos previos mediante llamada al método login()

\* @return "error" si no se puede validar el login

\* "success" en caso de que el login se haya validado

\* correctametne

\* @param connectionImpl Objeto que se debe validar

\*/

protected String validarLogin(ConnectionImpl connectionImpl);

/\*\*

\* Establece una conexión física con la base de datos a partir de

\* los

\* datos recibidos desde el formulario HTML en las propiedades

\* pertinentes

\* y añade el objeto a la sesión web del usuario

\* @return "error" si no se puede hacer login

\* "success" en caso de que se haya realizado el login

\* correctamente

\*/

public String login()

/\*\*

\* Intenta conseguir la conexión física con la base de datos. Si

\* la consigue

\* la guarda en la sesión del usuario

\* @param url URL a la base de datos

\* @param usuario Usuario que se quiere conectar a la base de

\* datos

\* @param password Clave del usuario

\* @return "error" si no se consigue la conexión, "success" en

\* caso contrario

\*/

private String login(String url, String usuario, String password)

...

}

#### MenuAction

Hereda de LoginAction y de ella heredan todas las acciones que finalmente muestren el menú de usuario

public class MenuAction extends LoginAction implements SessionAware {

protected String ACTION\_ERROR = "ACTION\_ERROR";

protected String ACTION\_MESSAGE = "ACTION\_MESSAGE";

private List<Menu> menus;

protected String[] filtroArgumentos;

protected String[] filtroValores;

protected Map<String, Object> sesion;

protected void cierraRS(ResultSet rs, PreparedStatement ps)

/\*\*

\* Obtiene en una List<List> el resultset recibido. El primer

\* elemento de

\* la lista representa los nombres de los campos del resultset

\* subyacente

\* @param resultSet Resultset a convertir en List<List>

\* @param metodoLink Indica si el método que proporciona el

\* resultset debe

\* tener un link cuando sea presentado en la jsp (true) o no

\* (false)

\* @linkParametros Cuando se requiera incluir una columna de link

\* (metodoLink = true)

\* se deberán indicar en este argumento la lista de parámetros que

\* se deben

\* incluir en la primera columna de la lista con la información

\* requerida

\* para realizar el link correctamente.

\* @return

\* @throws SQLException

\*/

protected List<List> getListInfo(ResultSet resultSet, boolean

metodoLink, String[] linkParametros) throws SQLException,

ResultSetException

/\*\*

\* Obtiene en una List<List> el resultset recibido. Los elementos

\* de la lista

\* están organizados en forma de formulario, el primer elemento

\* contiene el

\* nombre del campo, el segundo contiene el valor. En principio

\* pensado para

\* resultset con un solo registro, aunque si tuviera más no

\* cambiaría la

\* organización de los datos.

\* @param resultSet Resultset a convertir en List<List>

\* @param metodoLink Indica si el método que proporciona el

\* resultset debe

\* tener un link cuando sea presentado en la jsp (true) o no

\* (false)

\* @param linkParametros Cuando se requiera incluir una columna de

\* link (metodoLink = true)

\* se deberán indicar en este argumento la lista de parámetros que

\* se deben

\* incluir en la primera columna de la lista con la información

\* requerida

\* para realizar el link correctamente.

\* @param argumentos Si este parametro es null, la primera columna

\* contendrá

\* la cedena de la forma

\* linkParmetro1=valor1&linkParametro2=valor2. Pero si

\* este parametro no es nulo la cadena será de la forma

\* [argumentos]=linkParametro1&

\* [valores]=valor1&[argumentos]=linkParametro2&[valores]=valor2

\* @param valores Contiene el nombre del parámetro bajo el que se

\* incluyen los

\* valores

\* @return

\* @throws SQLException

\*/

protected List<List> getListInfoReverse(ResultSet resultSet)

/\*\*

\* Obtiene en una List<List> el resultset recibido. El primer

\* elemento de

\* la lista representa los nombres de los campos del resultset

\* subyacente

\* @param resultSet Resultset a convertir en List<List>

\* @param metodoLink Indica si el método que proporciona el

\* resultset debe

\* tener un link cuando sea presentado en la jsp (true) o no

\* (false)

\* @param linkParametros Cuando se requiera incluir una columna de

\* link (metodoLink = true)

\* se deberán indicar en este argumento la lista de parámetros que

\* se deben

\* incluir en la primera columna de la lista con la información

\* requerida

\* para realizar el link correctamente.

\* @param argumentos Si este parametro es null, la primera columna

\* contendrá

\* la cedena de la forma

\* linkParmetro1=valor1&linkParametro2=valor2. Pero si

\* este parametro no es nulo la cadena será de la forma

\* [argumentos]=linkParametro1&

\* [valores]=valor1&[argumentos]=linkParametro2&[valores]=valor2

\* @param valores Contiene el nombre del parámetro bajo el que se

\* incluyen los

\* valores

\* @return

\* @throws SQLException

\*/

protected List<List> getListInfo(ResultSet resultSet, boolean

metodoLink, String[] linkParametros, String argumentos,

String valores) throws SQLException, ResultSetException

public List<Menu> getMenus()

...

}

#### DatabaseMetaDataAction

public class DatabaseMetaDataAction extends MenuAction implements Preparable, SessionAware {

private ConnectionImpl connectionImpl;

private List<PropiedadValor> listInicial;

private String metodo;

private List<List> listInfo;

private String parametros;

Parameter[] arrayParametros; // parametros del método invocado

private String linkAction;

private String linkNamespace;

private int linkColumnNumber;

private boolean metodoLink;

private String[] linkParametros;

public Object getParameter(int i)

public List getArrayParametros()

/\*\*

\* Invoca el método de DatabaseMetaData con los parámetros

\* adecuados y forma la

\* lista con la información a presentar

\* @return "success" si no se produce error, "error" en caso

\* contrario

\*/

public String info()

/\*\*

\* Invoca el método de DatabaseMetaData con los parámetros

\* adecuados y forma la

\* lista con la información a presentar

\* @return "success" si no se produce error, "error" en caso

\* contrario

\*/

public String resultset()

/\*\* Obtenidos los tipos de los parámetros, se convierten los

\* valores

\* recibidos a los tipos correspondientes y se ponen en la sesión

\* para

\* sucesivas llamadas a este o a otros métodos que usen el mismo

\* parámetro (con el mismo nombre)

\* Convierte cada valor de filtroValores al tipo necesario según

\* se indica

\* en parameterTypes y lo añade a la sesión del usuario con el

\* nombre

\* correspondiente en filtroArgumentos

\*

\* @param parameterTypes Array de clases de los argumentos del

\* método que

\* se va a invocar en un momento posterior. Este array marca el

\* orden en el

\* que se procesan los elementos de filtroArgumentos y de

\* filtroValores

\*/

private void setParametrosSesion(Class[] parameterTypes)

private Class[] getParameterTypes(String parametros) throws

IOException, ClassNotFoundException

/\*\*

\* Ejecuta el método del objeto databaseMetaData indicado.

\* Devuelve el resultset

\* resultado de la ejecución.

\* @param databaseMetaData

\* @param metodo Nombre del método a invocar

\* @param parametros Array de clases que conforman los parámetros

\* del método

\* codificado en Base64

\* @return

\* @throws IllegalAccessException

\* @throws IllegalArgumentException

\* @throws NoSuchMethodException

\* @throws InvocationTargetException

\* @throws DatabaseMetaDataException Si el resultado de la

\* ejecución del método

\* no produce un objeto de tipo ResultSet se genera una excepción

\* de este tipo

\*/

private Object getInvoke(DatabaseMetaData databaseMetaData,

String metodo, Class[] parameterTypes)

throws IllegalAccessException, IllegalArgumentException,

NoSuchMethodException, InvocationTargetException,

DatabaseMetaDataException, IOException,

ClassNotFoundException

/\*\*

\* Se valida la conexión obtenida. Si error, se pasará a login.jsp

\*/

@Override

public void validate()

public String inicial()

/\*\*

\* Se obtiene la conexión de la sesión del usuario

\*

\* @throws Exception

\*/

@Override

public void prepare() throws Exception

...

}

#### TablasAction

public class TablasAction extends MenuAction implements Preparable, SessionAware {

private ConnectionImpl connectionImpl;

private Map<String, Object> sesion;

private String TABLE\_SCHEM;

private String TABLE\_NAME;

private String TABLE\_TYPE;

private List<List> listInfo;

private Set<String> arrayParametros;

private String[] pkArgumentos;

private String[] pkValores;

private boolean metodoLink;

private String[] formCampos;

private String[] formValores;

private Map<String, Integer> mapaEditables;

private Map<String, Integer[]> mapaCompleto;

public boolean isEditDisabled(String campo)

public String insertarGuardar()

public String insertar()

public Object getPkValores(int i)

public String borrar()

public Object getParameter(int i)

/\*\* Obtenidos los tipos de los parámetros, se convierten los

\* valores

\* recibidos a los tipos correspondientes y se ponen en la sesión

\* para

\* sucesivas llamadas a este método

\* Convierte cada valor de filtroValores al tipo necesario según

\* se indica

\* en el mapa obtenido de la sesion y grabado previamente, en la

\* primera

\* ejecución y lo añade a la sesión del usuario con el nombre

\* correspondiente en filtroArgumentos

\*

\* @param tabla nombre del esquema y la tabla. Se buscará en la

\* sesión un objeto map

\* con este nombre. Contendrá un mapa de los campos del resultset

\* y su tipo

\* @return mapa del resultset de la tabla con

\* @throws SQLException

\*/

private Map<String, Integer> setParametrosSesion(String tabla)

/\*\*

\* Consulta la tabla recibida en TABLE\_NAME y en TABLE\_SCHEM y la

\* presenta

\* en jsp en forma de lista

\* @return

\*/

public String consulta()

/\*\*

\* Se valida la conexión obtenida. Si error, se pasará a login.jsp

\*/

@Override

public void validate()

@Override

public void setSession(Map<String, Object> map)

/\*\*

\* Se obtiene la conexión de la sesión del usuario

\*

\* @throws Exception

\*/

@Override

public void prepare() throws Exception

}

## Diseño procedimental

A continuación describiremos cada uno de los subsistemas que se han presentado. Realizaremos una descripción de la funcionalidad del sistema mediante la vista de casos de usos, una descripción del modelo de datos que soporta.

### Conexión a la base de datos

Para un usuario nuevo, una solicitud a la página principal de la aplicación web (index.html) redirige a la acción pertinente que presentará al usuario la página donde deberá introducir los datos necesarios para establecer la conexión:

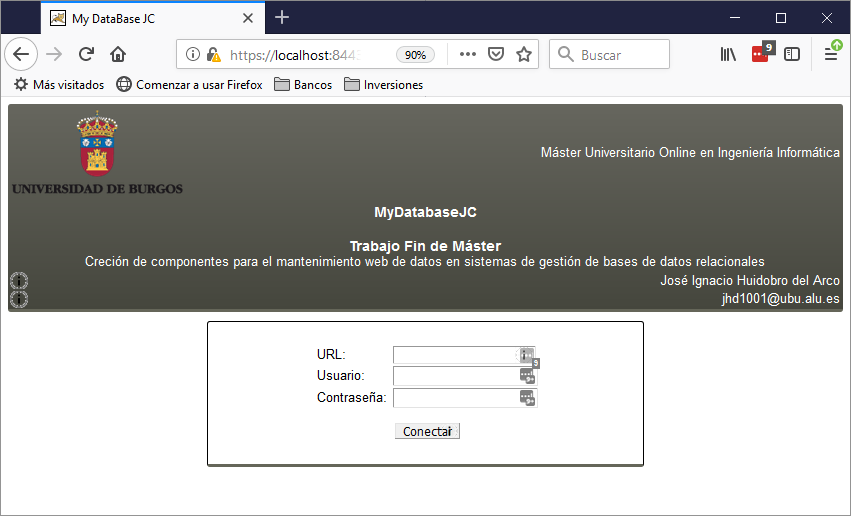


Figura 1: Pantalla de conexión con la base de datos

El proceso que recibe la request con los datos solicitados debe analizar la URL y en función de ella realizar la carga del driver JDBC correspondiente a la base de datos a la que se quiere conectar. Una vez producida la carga del driver, el proceso deberá intentar realizar una conexión física con la base de datos según los datos recibidos.

Si el intento de conexión es válido, el proceso instanciará un objeto de la clase ConnectionImpl que almacenará en la sesión del usuario mantenida por el servidor de aplicaciones web.

| Escenario de casos de uso Obtener conexión | |
| --- | --- |
| Caso de uso | Obtener conexión |
| Actores | Usuario, Base de datos |
| Descripción | Para que el usuario realice una conexión debe enviar los datos de conexión. La base de datos debe validar los datos de conexión |
| Precondición | 1. El usuario debe conocer los datos de conexión 2. La aplicación debe disponer del driver JDBC apropiado para la base de datos a la que se quiere conectar |
| Flujo normal | 1. El usuario rellena el formulario de conexión 2. El sistema decide que driver debe cargar en función de la URL 3. El sistema carga el driver JDBC 4. El sistema solicta una conexión a la base de datos con los datos recibidos 5. La base de datos acepta la solicitud y retorna una conexión válida a la aplicación 6. La aplicación almacena en la sesión del usuario la conexión y los datos de conexión para su uso entre llamadas por parte del mismo usuario |
| Flujo alterno | 1. La URL no informa de un driver posible    1. El sistema analiza la URL proporcionada y en función de su contenido determina qué driver JDBC debe ser cargado.    2. El sistema informa al usuario de que la URL no proporciona un driver disponible    3. El usuario se manteine en la pantalla de login 2. Error al cargar el driver    1. El sistema intenta cargar el driver determinado por la URL    2. El sistema no puede cargar el driver    3. Se informa al usuario del error al cargar el driver    4. El usuario se mantiene en la pantalla de login 3. Error estableciendo conexión    1. El sistema intenta establecer una conexión    2. La base de datos informa de un error al sistema en el intento de conexión    3. El sistema informa al usuario del error en el intento de conexión    4. El usuario se mantiene en la pantalla de login |
| Postcondición | El usuario ha realizado una conexión a la base de datos |

Tabla 29: Caso de uso Obtener conexión

Cada vez que el usuario realice una solicitud a la aplicación que requiera la utilización de la conexión a la base de datos, el sistema debe realizar una comprobación de varias situaciones necesarias para que la conexión pueda utilizarse.

| Escenario de casos de uso Utilizar conexión | |
| --- | --- |
| Caso de uso | Utilizar conexión |
| Actores | Usuario, Base de datos |
| Descripción | Para que el usuario utilice una conexión previamente establecida, se deben realizar una serie de pasos que se indican a continuación |
| Precondición | 1. El usuario debe haber establecido previamente una conexión 2. La conexión debe haber sido almacenada en la sesión del usuario en el servidor web |
| Flujo normal | 1. El usuario solicita una función que requiere la utilización de una conexión 2. La aplicación confirma que el usaario tiene en la sesión un objeto con una conexión 3. La aplicación solicita a la base de datos que la conexión sea válida y que esté abierta 4. La base de datos valida e informa de que la conexión es correcta y se encuentra abiert 5. La aplicación proporciona al proceso la conexión solicitada |
| Flujo alterno | 1. Si el sistema detecta que el usuario no dispone de una sesión válida, se retorna al usuario a la pantalla de login y se cancela el proceso original 2. Si el sistema detecta que el usuario no dispone en la sesión de un objeto de la clase ConnectionImpl, se retorna al usuario a la pantalla de login y se cancela el proceso original 3. Si el sistema detecta que la conexión no es válida o no está abierta, se retorna al usuario a la pantalla de login y se cancela el proceso original. |
| Postcondición | El sistema proporciona la conexión con la base de datos |

Tabla 30: Caso de uso Utilizar conexión

### Consulta de información de conexión

El usuario, una vez obtenida la conexión, recibirá la información de la conexión mostrando las propiedades más importantes de la misma.

Para obtener dicha información, se ejecutarán todos los métodos de la clase ConnectionInfo y se mostrará la información obtenida en forma de tabla html

| Escenario de casos de uso Información de la conexión | |
| --- | --- |
| Caso de uso | Información de la conexión |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la información de la conexión y se le muestra en pantalla |
| Precondición | 1. El usuario obtiene una conexión válida y activa |
| Flujo normal | 1. El usuario solicta la información de la conexión 2. El sistema instancia un objeto de la clase ConnectionInfo 3. El sistema obtiene una lista de los métodos de la clase 4. Para cada método, el sistema ejecuta el método y almacena el resultado 5. El sistema proporciona al usuario la lista de métodos y valores obtenidos en su ejecución |
| Flujo alterno |  |
| Postcondición |  |

Tabla 31: Caso de uso Información de la conexión

### Métodos de los metadata de la base de datos

Para que el usuario pueda realizar todas las operaciones requeridas con la base de datos, se le proporcionará una lista con los métodos disponibles relacionados con la base de datos. Existen una gran cantidad de métodos para obtener información de la báse de datos, aunque solo nos centraremos en dos tipos:

* Métodos que, con o sin parámetros, devuelven información en forma de resultset
* Métodos que, sin parámetros, devuelven información simple, un solo dato.

Para realizar esta funcionalidad, se crea una clase DatabaseMetaDataImpl que implementa todos los métodos del interface DatabaseMetaData y además implementa una serie de métodos que engloban en categorías aquellos métodos que, sin parámetros, devuelven datos simples. Estos métodos implementados retornarán la información en forma de lista de objetos PropiedadValor. Por lo tanto, finalmente, el sistema solo deberá contamplar de la clase DatabaseMetaDataImpl:

* Métodos que, con o sin parámetros, devuelven información en forma de resultset
* Métodos que, sin parámetros, devuelven información en forma de lista de PropiedadValor

| Escenario de casos de uso Obtener métodos del Metadata de la base de datos | |
| --- | --- |
| Caso de uso | Obtener métodos del metadata de la base de datos |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la lista de métodos. El sistema proporciona la lista de métodos junto con los tipos de parámetros necesarios para su ejecución |
| Precondición | 1. El usuario obtiene una conexión válida y activa |
| Flujo normal | 1. El usuario solicta la información de la base de datos 2. El sistema instancia un objeto de la clase DatabaseMetaDataImpl 3. El sistema recopila la lista de métodos de la clase 4. Para cada método, almacena nombre y clases de los argumentos si el tipo devuelto es resultset o list 5. El sistema proporciona al usuario la lista de métodos con sus nombres y la lista de tipos de sus argumentos |
| Flujo alterno |  |
| Postcondición | Para conseguir enviar al usuario la lista de los tipos de argumentos, esta debe ser convertida a un outputstream y posteriormente codificada en base64 puesto que la información va a viajar en la response y posteriormente volverá a la aplicación en una request. |

Tabla 32: Caso de uso Obtener métodos del Metadata de la base de datos

Java Reflection es un módulo integrado en el J2EE que permite, entre otras cosas, la ejecución de un método de una clase y la recepción del resultado en el objeto adecuado. Una vez que el usuario dispone de los métodos proporcionados por JDBC para consulta de bases de datos, el sistema le debe proporcionar la posibilidad de ejecutarlos adecuadamente y recoger y devolver los resultados al usuario.

| Escenario de casos de uso Ejecutar un método del metadata de la base de datos | |
| --- | --- |
| Caso de uso | Ejecutar un método del metadata de la base de datos |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la ejecución de un determinado método de los metadatos de la base de datos proporcionados por la clase DatabaseMetaDataImpl. El sistema le presenta los resultados obtenidos. Opcionalmente, el usuario puede añadir a la request uno o más argumentos para la ejecución del método. El nombre del parámetro debe coincidir con el nombre del argumento para que su asociación tenga validez. |
| Precondición | 1. El usuario obtiene una conexión válida y activa 2. El usuario dispone del nombre del método y de los tipos de sus argumentos así como de los valores que debe asignar a cada uno 3. La lista de tipos de los argumentos debe estar codificada en base64 para que le lleguen correctamente al sistema y pueda, a partir de ahí, obtener los tipos de los argumentos en el formato adecuado. |
| Flujo normal | 1. El usuario envía al sistema una request en la que se indica el método y los tipos de los argumentos que debe ejecutarse 2. El sistema averigua los nombres de los argumentos del método que el usuario desea ejecutar 3. Si el usuario incluye en la request algún parámetro que corresponda con un argumento de la ejecución del método, el sistema lo convierte al tipo adecuado y lo almacena en la sesión del usuario. 4. El sistema busca en la sesión del usuario los datos de los argumentos por su nombre. 5. El sistema invoca al método pasándole los valores de los argumentos necesarios 6. El sistema obtiene el resultado de la invocación (resultset o List) y lo presenta al usuario. |
| Flujo alterno | 1. Los tipos de argumentos pasados al sistema no se corresponden con los que el método requiere. Se presentará al usuario el error correspondiente 2. Los valores de los argumentos pasados al sistema en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente 3. El número de datos obtenidos excede el límite establecido. Se presenta el mensaje al usuario. |
| Postcondición | Los datos son presentados al usuario en forma de List<List> que permite recorrerlos mediante un doble bucle y presentarlos en forma de tabla HTML |

Tabla 33: Caso de uso Ejecutar un método del metadata de la base de datos

### Trabajo con tablas

Una de las operaciones más importantes de la aplicación debe ser, como no podía ser de otra manera, la posibilidad de consultar, modificar, borrar o insertar registros en las tablas de la base de datos, puesto que, al fin y al cabo, esta es la utilidad última de una base de datos.

Para realizar estas operaciones, los sistemas de bases de datos relaciones disponen de un lenguaje de consulta y manipulación de datos en tablas denominado SQL.

La aplicación debe proporcionar los mecanismos necesarios para que el usuario pueda lanzar los comandos SQL más importantes:

* SELECT
* DELETE
* UPDATE
* INSERT

En todos estos comandos debe indicarse el nombre de la tabla sobre la que se debe realizar la operación. Puede ir o no acompañada del nombre del esquema propietario de la tabla.

En algunos casos se debe incluir la colección de campos/valor que permiten filtrar o acotar el ámbito de datos de ejecución del comando, y en otros se debe incluir además, los datos nuevos que deben ser almacenados en la base de datos, bien en una operación de actualización (UPDATE) bien en una operación de inserción (INSERT).

Para componer una correcta instrucción SQL se crea la clase SQLCommand que se explica en el diccionario de datos.

A continuación se indican los distintos casos de uso dentro de esta funcionalidad que corresponden con las operaciones básicas sobre datos solicitadas en los requerimientos

| Escenario de casos de uso Consultar datos de una tabla | |
| --- | --- |
| Caso de uso | Consultar datos de una tabla |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la consulta de los datos de una tabla. El sistema solicita a la base de datos la consulta de los datos de la tabla y devuelve al usuario los datos obtenidos |
| Precondición | 1. El usuario obtiene una conexión válida y activa 2. El usuario dispone del nombre de la tabla y, opcionalmente, la colección de campos y valores que representan el filtro a aplicar en la colección de datos |
| Flujo normal | 1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla 3. Opcionalmente, el usuario puede indicar uno o varios campos y sus correspondientes valores para conformar la matriz de filtrado de datos 4. El sistema convierte los datos recibidos como parte del filtro en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema guarda en la sesión del usuario los valores recibidos como parámetros de filtrado para que el usuario pueda reutilizarlos en consultas posteriores. 6. El sistema compone la consulta en lenguaje SQL estándar 7. El sistema traslada la consulta SQL a la base de datos 8. La base de datos comienza la transferencia de datos en forma de ResultSet 9. El sistema convierte los datos a formato List<List> y los entrega al usuario |
| Flujo alterno | 1. Alguno de los datos enviados por el usuario no forman parte de los posibles campos de consulta de la tabla. Se presenta el correspondiente error al usuario. 2. Los valores de los campos pasados al sistema como parámetros del filtro en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente 3. El número de datos obtenidos excede el límite establecido. Se presenta el mensaje al usuario. |
| Postcondición | Los datos son presentados al usuario en forma de List<List> que permite recorrerlos mediante un doble bucle y presentarlos en forma de tabla HTML |

Tabla 34: Caso de uso Consultar datos de una tabla

| Escenario de casos de uso Eliminación del registro de una tabla | |
| --- | --- |
| Caso de uso | Eliminar un registro de una tabla |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la eliminación del registro de una tabla. El sistema solicita a la base de datos la eliminación del registro de la tabla y devuelve al usuario el resultado de la operación |
| Precondición | 1. El usuario obtiene una conexión válida y activa 2. El usuario dispone del nombre de la tabla 3. El usuario dispone del nombre y valor de los campos que forman la primary key del registro que se quiere eliminar |
| Flujo normal | 1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla 3. El usuario debe indicar, obligatoriamente, los campos que conforman la primary key de la tabla y los valores para ellos que localizan unívocamente al registro a eliminar. 4. El sistema convierte los datos recibidos como parte del filtro (la primary key) en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema compone la consulta en lenguaje SQL estándar 6. El sistema traslada la consulta SQL a la base de datos 7. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación 8. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno |
| Flujo alterno | 1. Alguno de los datos enviados por el usuario no forman parte de los posibles campos de la primary key de la tabla. Se presenta el correspondiente error al usuario. 2. Los valores de los campos pasados al sistema como parámetros de la primary key en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente |
| Postcondición |  |

Tabla 35: Caso de uso Eliminación del registro de una tabla

| Escenario de casos de uso Actualización de los datos del registro de una tabla | |
| --- | --- |
| Caso de uso | Actualización de los datos del registro de una tabla |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la modificación de un registro de una tabla. El sistema solicita a la base de datos la modificación del registro de la tabla y devuelve al usuario el resultado de la operación |
| Precondición | 1. El usuario obtiene una conexión válida y activa 2. El usuario dispone del nombre de la tabla 3. El usuario dispone del nombre y valor de los campos que forman la primary key del registro que se quiere eliminar 4. El usuario dispone de los nombres de los campos y los valores que desea modificar |
| Flujo normal | 1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla 3. El usuario debe indicar, obligatoriamente, los campos que conforman la primary key de la tabla y los valores para ellos que localizan unívocamente al registro a eliminar. 4. El usuario debe indicar, obligatoriamente, los campos que se desean modificar junto a los nuevos valores para dichos campos 5. El sistema convierte los datos recibidos como parte del filtro (la primary key) y del conjunto de campos que se desean modificar, en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 6. El sistema compone la consulta en lenguaje SQL estándar 7. El sistema traslada la consulta SQL a la base de datos 8. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación 9. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno |
| Flujo alterno | 1. Alguno de los datos enviados por el usuario no forman parte de los posibles campos de la primary key o del conjunto de campos modificables de la tabla. Se presenta el correspondiente error al usuario. 2. Los valores de los campos pasados al sistema como parámetros de la primary key o del conjunto de campos a modificar en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente |
| Postcondición |  |

Tabla 36: Caso de uso Actualización de los datos del registro de una tabla

| Escenario de casos de uso Inserción de un registro de una tabla | |
| --- | --- |
| Caso de uso | Inserción del registro de una tabla |
| Actores | Usuario, Base de datos |
| Descripción | El usuario solicita la inserción de un registro de una tabla. El sistema solicita a la base de datos la inserción del registro de la tabla y devuelve al usuario el resultado de la operación |
| Precondición | 1. El usuario obtiene una conexión válida y activa 2. El usuario dispone del nombre de la tabla 3. El usuario dispone de los nombres de los campos y los valores que desea modificar |
| Flujo normal | 1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla 3. El usuario debe indicar, obligatoriamente, los campos que se desean insertar junto a los nuevos valores para dichos campos en un nuevo registro 4. El sistema convierte los datos recibidos como parte del conjunto de campos que se desean insertar, en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema compone la consulta en lenguaje SQL estándar 6. El sistema traslada la consulta SQL a la base de datos 7. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación 8. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno |
| Flujo alterno | 1. Alguno de los datos enviados por el usuario no forman parte de los posibles campos de la primary key o del conjunto de campos modificables de la tabla. Se presenta el correspondiente error al usuario. 2. Los valores de los campos pasados al sistema del conjunto de campos a modificar en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente |
| Postcondición |  |

Tabla 37: Caso de uso Inserción de un registro de una tabla

## Diseño arquitectónico

A continuación se describe el diseño de la aplicación en cuanto al flujo del programa atendiendo a cada uno de los casos de uso explicados. Para ello nos ayudaremos de un conjunto de gráficos y explicaciones junto con las correspondientes llamadas e instanciaciones de clases necesarias para realizar las tareas encomendadas

### Conexión a la base de datos

#### Obtener conexión

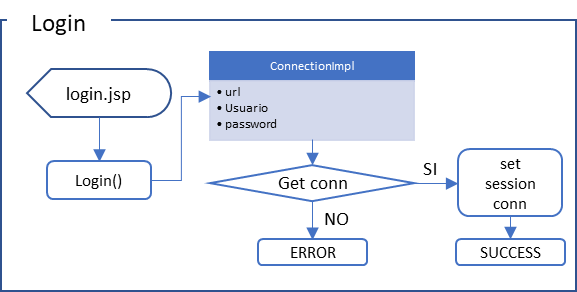


Figura 2 : Obtener conexión

En login.jsp, el formulario html recoje la información del usuario para establecer la conexión.

Mapeado por struts.xml, la ejecución continúa en el método login() de la clase LoginAction que instancia un objeto de ConnectionImpl y establece la conexión.

Una vez establecida la conexión se almacena el objeto ConnectionImpl en la sesión del usuario para poderse recuperar entre request distintas.

#### Utilizar conexión

Siempre que un usuario necesita una conexión con la base de datos se debe seguir el siguiente método para que el sistema se la proporcione:

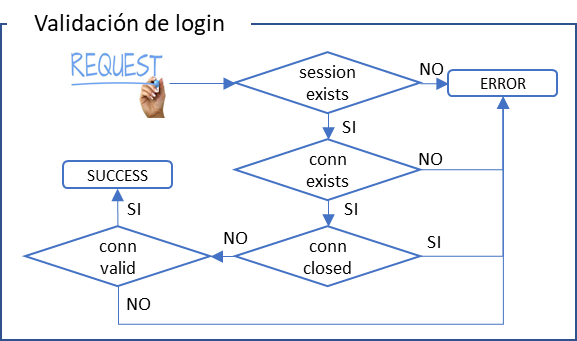


Figura 3: Utilización de login

En primer lugar el sistema comprueba que la sesión del usuario exista. Para ello se llama al método validarLogin() de la clase LoginAction. Esta llamada se debe realizar en cada petición de conexión, por lo que la clase LoginAction es la clase padre de todas las clases actions de la aplicación y en todas ellas existe una llamada a dicho método a través de el método validate() que debe estar en cada clase action y en LoginAction.

### Consulta de información de conexión

#### Información de la conexión

De forma automática, una vez que se inicia la aplicación y después de obtenida la conexión, se presenta al usuario la información general de la conexión mediante la llamada a una serie de métodos del objeto instanciado de la clase ConnectionInfo

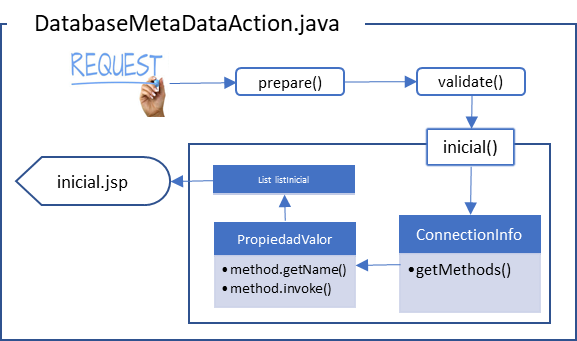


Figura 4: Presentación inicial de pantalla

El foco de ejecución del programa pasa al método inicial() de la clase DatabaseMetaDataAction que instancia un objeto de la clase ConnectionInfo y mediante java Reflection ejecuta los métodos en ella definidos y compone, junto a los nombres de esos métodos, con los resultados obtenidos de su ejecución una lista de objetos de la clas PropiedadValor que posteriormente son presentados al usuario en la página inicial.jsp.

En inicial.jsp, se recorre la lista de información presentándola al usuario.

### Métodos de los metadata de la base de datos

#### Obtener métodos del Metadata de la base de datos

Constituyen lo que viene a ser el menú de trabajo de la aplicación. Al usuario se le presentan los distintos métodos que nos interesan del objeto DatabaseMetaData de la conexión para lo que implementamos nuestra propia clase DatabaseMetaDataImpl que utilizaremos para obtener estos métodos. El sistema obtiene la lista de métodos con la información necesaria de ellos para, en una fase posterior, poder realizar una invocación de estos métodos y obtener su resultado y presentarlo al usuario.

La lista de menús se obtiene del método getMenus() de la clase MenuAction, que hereda directamente de LoginAction y de la que heredan todas las acciones en las que sea necesario manejar el menú, como por ejemplo DatabaseMetaDataAction.

En el fichero menu.jsp se procede a la representación de las opciones del menú definiendo los links para visualizar los resultados de segui cada una de las opciones del menú:

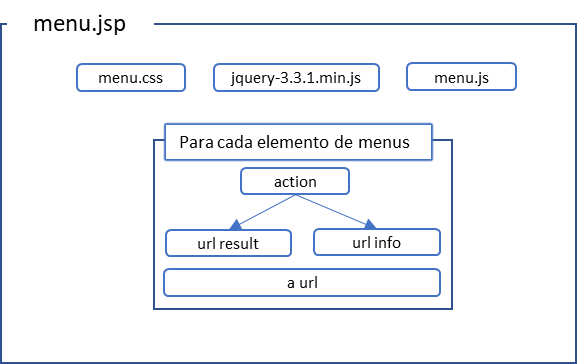


Figura 5: menu.jsp

#### Ejecutar un método del metadata de la base de datos

Existen dos métodos que permiten ejecutar cualquier método del metadata de la base de datos desde el menú del usuario. Su comportamiento es básicamente el mismo y difieren en la fase final, en la presentación de la información al usuario. El método resultset() de DatabaseMetaDataAction, al final, debe realizar una conversión de los datos desde un objeto de clase ResultSet a un objeto de clase List<List>. Además, el método debe tener en cuenta que para invocar los métodos de este tipo, deben proporcionarse los valores de los argumentos del tipo adecuado. El método info() de la misma clase difiere en que el resultado ya está en forma List<List> y no es necesario proporcionar valores en la invocación de los métodos.

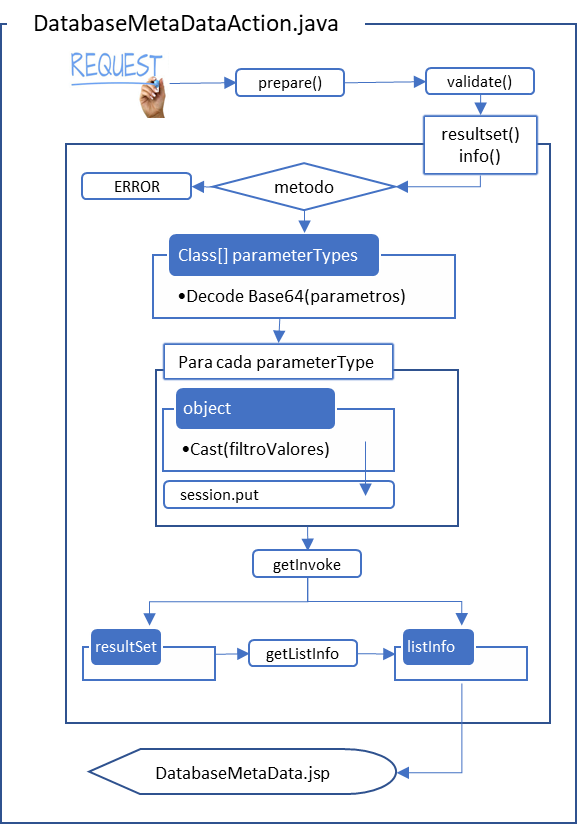


Figura 6: Ejecutar un método del metadata de una base de datos

Es interesante seguirle la pista al método getParameterTypes que recibe los parámetros de la request y los convierte en un array de clases.

También es interesante el método setParametrosSesion que convierte los parámetros recibidos de la request, los convierte al tipo adecuado y los guarda en la sesión del usuario para su utilización posterior.

A continuación se ejecuta el método getInvoke() que obtiene, bien un resultset, bien un List<List> en función del método requerido.

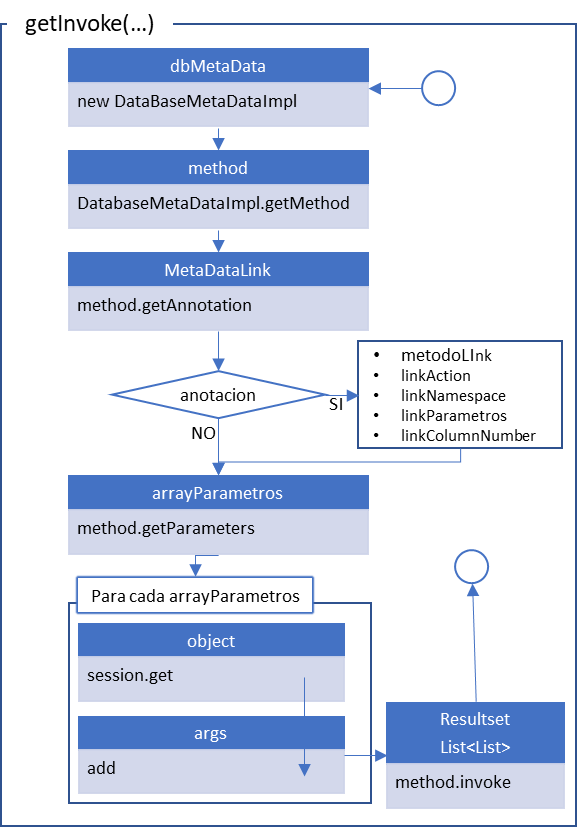


Figura 7: getInoke()

Finalmente, considerar el método getListInfo() de la clase MenuAction que, a partir de un ResultSet, devuelve una lista de listas (List<List>) para su posterior presentación al usuarios.

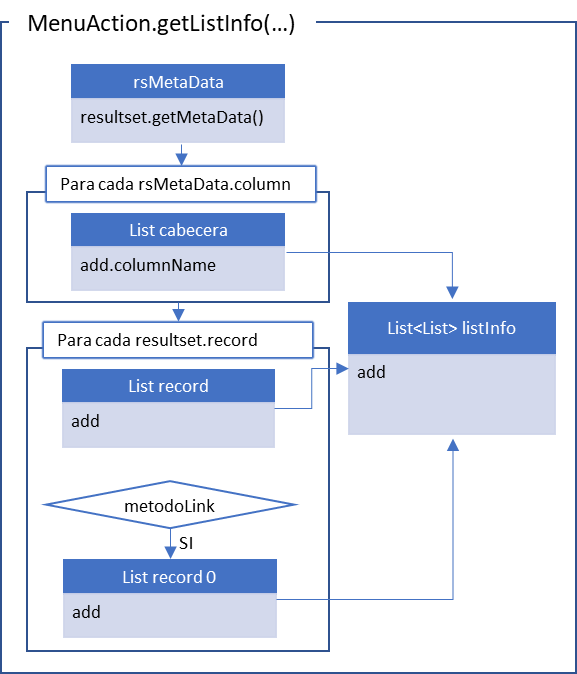


Figura 8: getListInfo()

Como paso final, solo queda presentar la información al usuario mediante un doble bucle que recorre la estructura List<List> incluyendo, si fuera necesario, elementos de link obtenidos a partir de la utilización de la anotación MetaDataLink en los métodos así requeridos de los metadata de la base de datos (en el trabajo solo se ha utilizado en el método getTables()). Tener en cuenta el tratamiento diferenciado de la primera fila como fila de títulos de la tabla html.

También es interesante indicar como se construye el formulario html que permitirá al usuario enviar al sistema los valores de los argumentos de invocación de los métodos. El código se encuentra en filtro.jsp y, utilizando un iterador, permite adaptarse a cualquier número de argumentos, variable en función del método a invocar. Los campos se organizan en dos columnas.

El objeto arrayParametros referenciado es establecido en el método getInvoke.

### Trabajo con tablas

#### Consultar datos de una tabla

Esta sería la primera funcionalidad que se debería implementar, para presentar al usuario los registros filtrados de la tabla deseada. Los comandos de consulta en bases de datos relacionales utilizan el lenguaje SQL, y el comando particular de consulta es la sentencia SELECT. Esta, y todas las demás desarrolladas en este trabajo, están agrupadas en una clase diseñada a tal efecto denominada SQLCommand. Cualquier operación comenzará con la instanciación de un objeto de esta clase para lo cual es necesario pasar una conexión JDBC válida.

Después, solo deberemos invocar al método adecuado con los datos pertinentes para realizar la operación contra la base de datos oportuna.

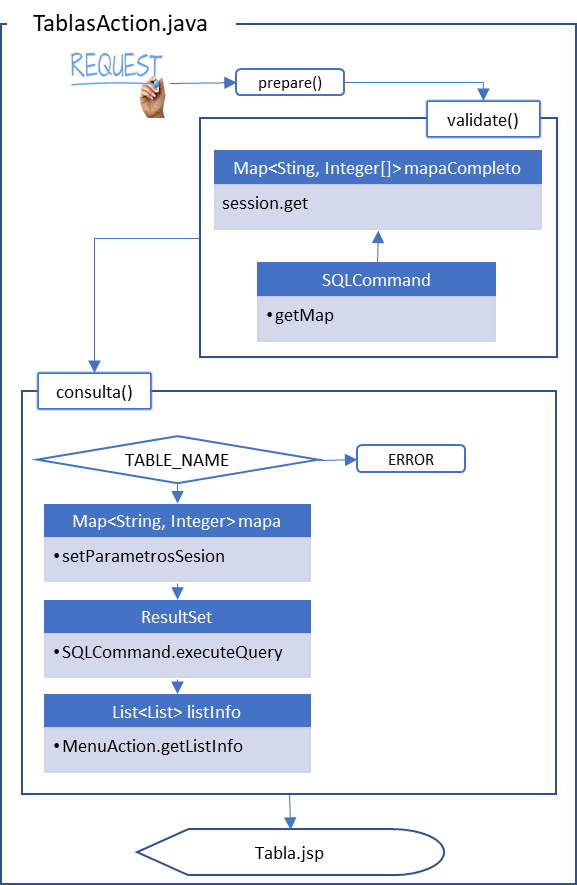


Figura 9: TablasAction. consulta()

Como se puede ver, el mapeado de la request lleva a la aplicación a ejecutar el método consulta() de la clase TablasAction, pero antes, se incluye en la validación el control de los datos necesarios para una correcta llamada posterior. En el método validate() se instancia un mapa con todos los campos de la tabla a consultar y se guarda en la sesión para no tener que realizarlo más veces para esa tabla.

A continuación, el método consulta() insta a la ejecución del comando SQL y al final carga una List<List> utilizando el método ya explicado getListInfo.

En la pantalla tabla.jsp se presenta al usuario el conjunto de datos de una forma similar a la explicada en DatabaseMetaData.jsp. Solo comentar que, con el fin de enlazar la consulta con las operaciones de eliminación, edición o inserción, se incluyen a modo de ejemplo.

En la clase SQLCommand, el método executeQuery nos permite obtener el resultset de la tabla.

Que nos lleva al método setSQLPartList para informar las variables oportunas.

Y al método executeQuery con los parámetros adecuados.

Finalmente, mostrar el formulario html que le permite al usuario aportar valores para realizar los filtros adecuados mediante el mismo código ya utilizado para la invocación de métodos del metadata de la base de datos.

#### Eliminar un registro de una tabla

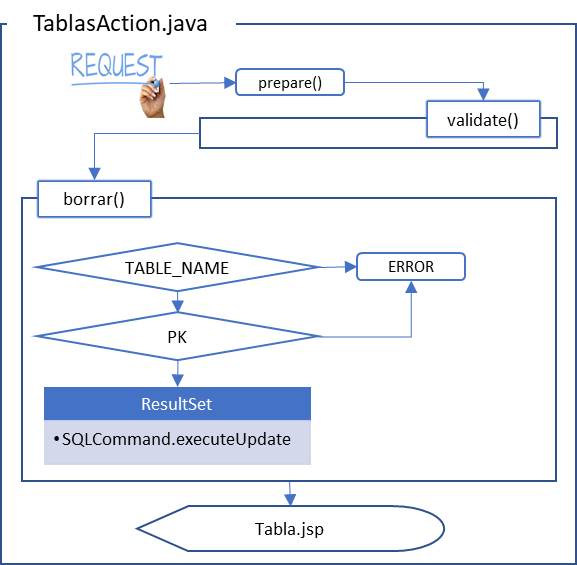


Figura 10: borrar()

Con una llamada a SQLCommand.executeUpdate que se comentará en el punto siguiente

#### Editar un registro de una tabla

Con un paso a través de un formulario en el que se presentan los campos del registro, el usuario podrá en dicho formulario modificarlos y retornarlos al sistema para que se lance el correspondiente comando a la base de datos, teniendo en cuanta que necesitamos no solo los datos que han de ser modificados sino también los campos que conforman la primary key de la tabla.

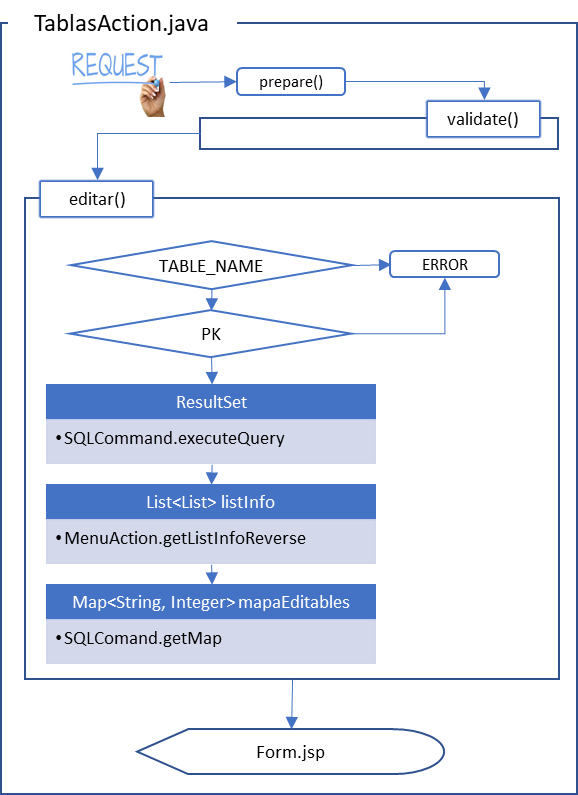


Figura 11: editar()

El proceso siguiente es el encargado de obtener los datos del formulario y guardarlos en la base de datos

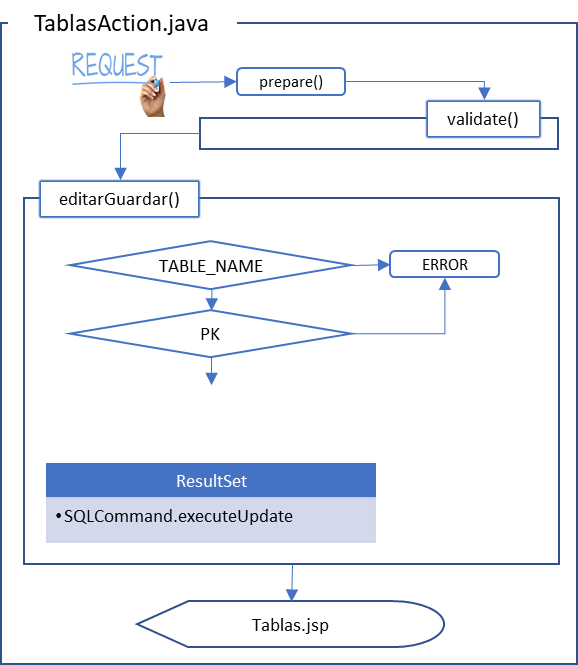


Figura 12editarGuardar()

Que llama a la calse executeUpdate de SQLCommand.

Y este a su vez hace dos llamadas al setSQLPartList.

#### Inserción de un registro en una tabla

De forma análoga se realizaría la funcionalidad de inserción de datos en una tabla de la base de datos.

Apéndice D

# Documentación técnica de programación

## Introducción

Como ampliación de la sección anterior, desarrollamos el siguiente capítulo indicando la estructura de directorios, los detalles de programación más relevantes atendiendo a los casos de uso explicados, las instrucciones de compilación, instalación y ejecución del proyecto y detallando las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación

## Estructura de directorios

### MyDataBaseJC/src/main

Carpeta raiz de códigos fuente

#### java

Carpeta con código fuente de clases java

#### resources

Carpeta con properties y ficheros de configuración xml

#### webapp

Estructura de carpetas para aplicación web. En ella incluimos los siguientes directorios

|  |  |
| --- | --- |
| Carpeta | Descripción |
| . | Se ubican aquí los ficheros jsp y html |
| META-INF | Ubicación de context.xml |
| WEB-INF | Ubicación de carpeta jspf con páginas jsp que se reutilizan en varias páginas jsp diferentes y web.xml |
| resources | Ubicación de ficheros de recursos: imágenes, hojas de estilo y javascript |

Tabla 38: Estructura de carpetas

### MyDataBaseJC/target

Carpeta raiz de soluciones compiladas. En ella nos encontramos una versión comprimida en formato war (myDatabaseJC-1.0-SNAPSHOT.war) junto con la aplicación en formato desplegado que puede ser automáticamente trasladado a un servidor de aplicaciones web bajo la carpeta myDataBaseJC-1.0-SNAPSHOT

## Manual del programador

A continuación se describe el diseño de la aplicación en cuanto al flujo del programa atendiendo a cada uno de los casos de uso explicados. En los procesos ya explicados en el apartado 3.4 Diseño arquitectónico, detallamos ahora las cuestiones técnicas relativas a la programación. Este capítulo es, por tanto, una ampliación del indicado anteriormente.

### Conexión a la base de datos

#### Obtener conexión

En login.jsp, el formulario html recoje la información del usuario para establecer la conexión:

<s:form action="login" method="POST" namespace="/"

id="filtro">

<s:textfield name="url" key="URL"

title='%{ayuda("URL")}' class="ayuda"/>

<s:textfield name="usuario" key="Usuario"/>

<s:password name="password" key="Contrasena"/>

<s:submit key="Conectar" name="conectar"

title='%{ayuda("Conectar")}' class="ayuda"/>

</s:form>

Mapeado por struts.xml, la ejecución continúa en el método login() de la clase LoginAction que instancia un objeto de ConnectionImpl y establece la conexión:

private String login(String url, String usuario,

String password) {

ConnectionImpl connectionImpl =

new ConnectionImpl(url, usuario, password);

HttpServletRequest request = ServletActionContext

.getRequest();

request.getSession()

.setAttribute(CONEXION, connectionImpl);

return SUCCESS;

}

Una vez establecida la conexión se almacena el objeto ConnectionImpl en la sesión del usuario para poderse recuperar entre request distintas.

#### Utilizar conexión

Siempre que un usuario necesita una conexión con la base de datos se debe seguir el siguiente método para que el sistema se la proporcione:

En primer lugar el sistema comprueba que la sesión del usuario exista. Para ello se llama al método validarLogin() de la clase LoginAction. Esta llamada se debe realizar en cada petición de conexión, por lo que la clase LoginAction es la clase padre de todas las clases actions de la aplicación y en todas ellas existe una llamada a dicho método a través de el método validate() que debe estar en cada clase action:

@Override

public void validate() {

super.validarLogin(connectionImpl);

}

Y en LoginAction:

public String validarLogin() {

// control de validez de sesión iniciada

HttpServletRequest request = ServletActionContext

.getRequest();

if (request.getSession().isNew()) return ERROR;

ConnectionImpl connectionImpl =

(ConnectionImpl)request.getSession()

.getAttribute(CONEXION);

return validarLogin(connectionImpl);

}

También en la misma clase:

protected String validarLogin(

ConnectionImpl connectionImpl) {

if (connectionImpl==null) {

addActionError(getText("Conexión no realizada"));

return ERROR;

}

// si la conexión se ha cerrado, se intenta

establecer nuevamente

if (connectionImpl.isClosed())

return login(

connectionImpl.getUrl(),

connectionImpl.getUsuario(),

connectionImpl.getPassword());

// si la conexión no es válida, se intenta establecer nuevamente

if (!connectionImpl.isValid(0))

return login(

connectionImpl.getUrl(),

connectionImpl.getUsuario(), connectionImpl.getPassword());

// validación de login correcta

return SUCCESS;

}

### Consulta de información de conexión

#### Información de la conexión

De forma automática, una vez que se inicia la aplicación y después de obtenida la conexión, se presenta al usuario la información general de la conexión mediante la llamada a una serie de métodos del objeto instanciado de la clase ConnectionInfo

El foco de ejecución del programa pasa al método inicial() de la clase DatabaseMetaDataAction que instancia un objeto de la clase ConnectionInfo y mediante java Reflection ejecuta los métodos en ella definidos y compone, junto a los nombres de esos métodos, con los resultados obtenidos de su ejecución una lista de objetos de la clas PropiedadValor que posteriormente son presentados al usuario en la página inicial.jsp:

public String inicial() {

// Modela la clase de información de la conexión

para su presentación

ConnectionInfo connectionInfo =

new ConnectionInfo(connectionImpl

.getConnection());

// inicia la lista de valores

listInicial = new ArrayList<>();

// obtiene los métodos de la clase

Method[] methods = ConnectionInfo.class

.getMethods();

// para cada método

for (Method method : methods) {

// solo de la clase base 8excluye los de

clase padre

if (method.getDeclaringClass().equals(

connectionInfo.getClass())) {

Object o = null;

try {

o = method.invoke(

connectionInfo,

new Object[]{});

} catch (Exception e) {

o = e.getLocalizedMessage();

}

// crea el metodo/valor. La propiedad es

el nombre del método

// y el valor es el resultado de la

ejecución del método

PropiedadValor p = new PropiedadValor(

method.getName(), o);

// y lo añade a la lista

listInicial.add(p);

}

}

return SUCCESS;

}

En inicial.jsp, se recorre la lista de información presentándola al usuario:

<s:iterator value="listInicial">

<tr>

<td title=

'<s:property value="%{ayuda(propiedad)}"/>'>

<s:property value="%{getText(propiedad)}"/>

</td>

<td><s:property value="valor"/></td>

</tr>

</s:iterator>

### Métodos de los metadata de la base de datos

#### Obtener métodos del Metadata de la base de datos

Constituyen lo que viene a ser el menú de trabajo de la aplicación. Al usuario se le presentan los distintos métodos que nos interesan del objeto DatabaseMetaData de la conexión para lo que implementamos nuestra propia clase DatabaseMetaDataImpl que utilizaremos para obtener estos métodos. El sistema obtiene la lista de métodos con la información necesaria de ellos para, en una fase posterior, poder realizar una invocación de estos métodos y obtener su resultado y presentarlo al usuario.

La lista de menús se obtiene del método getMenus() de la clase MenuAction, que hereda directamente de LoginAction y de la que heredan todas las acciones en las que sea necesario manejar el menú, como por ejemplo DatabaseMetaDataAction:

public List<Menu> getMenus() { //throws IOException {

// inicia la lista de valores del menu

menus = new ArrayList<>();

// A continuación obtiene los métodos de la MetaData que devuelven

// objetos de tipo ResultSet que serán añadidos

como opciones de menú

Method[] methods = DatabaseMetaDataImpl.class

.getMethods();

int index;

// para cada método

for (Method method : methods) {

// solo si el método retorna un tipo

ResultSet

if (method.getReturnType()==ResultSet.class

|| method.getReturnType()==List.class) {

// crea el menu.

Menu menu = new Menu(

method.getReturnType()==

ResultSet.class ? "resultset" :

"info",

method.getName(),

method.getParameterTypes());

// lo busca en la lista

if ((index =

menus.indexOf(menu))==-1) {

// y lo añade el nombre del

método a la lista si no existe

aún

menus.add(menu);

}

}

}

return menus;

}

En el fichero menu.jsp se procede a la representación de las opciones del menú definiendo los links para visualizar los resultados de segui cada una de las opciones del menú:

<s:iterator value="menus">

<li class="ayuda">

<s:url action="%{action}" var="urlTag"

namespace="/DatabaseMetaData">

<s:param name="metodo">

<s:property value="metodo"/>

</s:param>

<s:param name="parametros">

<s:property value="parametros"/>

</s:param>

</s:url>

<s:a href="%{urlTag}" title='%{ayuda(metodo)}'>

<s:property value="%{getText(metodo)}"/>

</s:a>

</li>

</s:iterator>

#### Ejecutar un método del metadata de la base de datos

Existen dos métodos que permiten ejecutar cualquier método del metadata de la base de datos desde el menú del usuario. Su comportamiento es básicamente el mismo y difieren en la fase final, en la presentación de la información al usuario. El método resultset() de DatabaseMetaDataAction, al final, debe realizar una conversión de los datos desde un objeto de clase ResultSet a un objeto de clase List<List>. Además, el método debe tener en cuenta que para invocar los métodos de este tipo, deben proporcionarse los valores de los argumentos del tipo adecuado. El método info() de la misma clase difiere en que el resultado ya está en forma List<List> y no es necesario proporcionar valores en la invocación de los métodos.

public String resultset() {

// Recoge los mensajes de error previos y los

establece en esta acción

if (sesion.get(ACTION\_ERROR)!=null) {

addActionError(getText((String)sesion

.get(ACTION\_ERROR)));

sesion.remove(ACTION\_ERROR);

}

// Si metodo no contiene ningún valor, error

if (metodo==null || metodo.length()==0) {

addActionError(

getText(

"Llamada.a.la.accion.resultset.incorrecta"));

return ERROR;

}

ResultSet rs = null;

try {

Class[] parameterTypes =

getParameterTypes(parametros);

setParametrosSesion(parameterTypes);

rs = (ResultSet)getInvoke(connectionImpl

.getConnection().getMetaData(),

metodo, parameterTypes);

listInfo = getListInfo(rs, metodoLink,

linkParametros);

} catch (ResultSetException | SQLException |

NoSuchMethodException | SecurityException |

IllegalAccessException |

IllegalArgumentException |

InvocationTargetException | IOException |

ClassNotFoundException |

DatabaseMetaDataException ex) {

try {if (rs!=null) rs.close();}

catch (SQLException ex1) {;}

addActionError(ex.getMessage());

return ERROR;

}

return SUCCESS;

}

Es interesante seguirle la pista al método getParameterTypes que recibe los parámetros de la request y los convierte en un array de clases:

private Class[] getParameterTypes(String parametros)

throws IOException, ClassNotFoundException {

// se decodifica y obtiene el array de bytes que

forman los parámetros del métod

ByteArrayInputStream bais =

new ByteArrayInputStream(Base64

.decodeBase64(parametros));

ObjectInputStream ois =

new ObjectInputStream(bais);

// se obtiene el array de parámetros a partir de

los datos obtenidos

return (Class[])ois.readObject();

}

También es interesante el método setParametrosSesion que convierte los parámetros recibidos de la request, los convierte al tipo adecuado y los guarda en la sesión del usuario para su utilización posterior:

private void setParametrosSesion(

Class[] parameterTypes) {

if (filtroArgumentos == null ||

filtroValores == null) return;

for (int i = 0; i < parameterTypes.length; i++) {

String atributo = filtroArgumentos[i];

Object valor = null;

if (!"".equals(filtroValores[i]))

switch (parameterTypes[i].getName()) {

case "[Ljava.lang.String;":

String[] valores = filtroValores[i]

.split(",",0);

List<String> lvalores = new ArrayList();

for (int j = 0; j< valores.length; j++)

if (!"".equals(valores[j]))

lvalores.add(valores[j].trim());

valor = lvalores.toArray(valores);

break;

case "boolean": valor = Boolean

.valueOf(filtroValores[i]); break;

case "int": valor = Integer

.valueOf(filtroValores[i]); break;

case "short": valor = Short

.valueOf(filtroValores[i]); break;

default: valor = filtroValores[i];

}

sesion.put(atributo, valor);

}

}

A continuación se ejecuta el método getInvoke() que obtiene, bien un resultset, bien un List<List> en función del método requerido.

private Object getInvoke(DatabaseMetaData databaseMetaData,

String metodo, Class[] parameterTypes)

throws IllegalAccessException, IllegalArgumentException,

NoSuchMethodException, InvocationTargetException,

DatabaseMetaDataException, IOException,

ClassNotFoundException {

// Obtiene el objeto DatabaseMetaDataImpl

DatabaseMetaDataImpl dbMetadata =

new DatabaseMetaDataImpl(databaseMetaData);

// Se invoca el método solicitado, con los parámetros

calculados

Method method = DatabaseMetaDataImpl.class.getMethod(

metodo, parameterTypes);

// obtiene la anotación del tipo MetaDataLink para determinar

si el método

// debe conllevar un link asociado a cada registro del

resultset

MetaDataLink anotacion = method

.getAnnotation(MetaDataLink.class);

metodoLink = false;

// si existe la anotación, se definen la acción, el namespace

y los parámetros

if (anotacion != null) {

metodoLink = true;

linkAction = anotacion.action();

linkNamespace = anotacion.namespace();

linkParametros = anotacion.parametros();

linkColumnNumber = anotacion.columnNumber();

}

// Obtiene los nombres de los parametros

arrayParametros = method.getParameters();

// crea una lista de objetos que serán finalmente los

parametros a pasar al metodo invocado

List<Object> listaParametros = new ArrayList<>();

// busca cada parametro en la sesión del usuario

for (Parameter parametro : arrayParametros) {

// y lo asigna a la lista de objetos a pasar

Object o = sesion.get(parametro.getName());

if (o!=null) if ("".equals(o.toString())) o = null;

listaParametros.add(o);

}

// Obtiene la lista de objetos en forma de array. Serán los

argumentos de

// la función invocada

Object[] args = listaParametros.toArray();

// e invoca al método pasandole los argumentos necesarios

Object o = method.invoke(dbMetadata, args);

if (o instanceof ResultSet) return (ResultSet)o;

if (o instanceof List) return (List<List>)o;

throw new DatabaseMetaDataException(

getText(

"El.objeto.obtenido.no.es.del.tipo.adecuado"), metodo);

}

Finalmente, considerar el método getListInfo() de la clase MenuAction que, a partir de un ResultSet, devuelve una lista de listas (List<List>) para su posterior presentación al usuarios.

protected List<List> getListInfo(ResultSet resultSet,

boolean metodoLink, String[] linkParametros,

String argumentos, String valores)

throws SQLException, ResultSetException {

List<List> lista = new ArrayList();

// se obtiene el metadata del result set

ResultSetMetaData rsMetadata = resultSet.getMetaData();

// y se almacenan los nombres de las columnas como primer

elemento

// de la lista

int i = 0;

List cabecera = new ArrayList();

// si es un método que debe llevar link, se añade un objeto

antes de los datos

// en cada registro. La primera columna tendrá los parámetros

del link

if (metodoLink) cabecera.add(null);

for (i = 1; i <= rsMetadata.getColumnCount(); i++) {

cabecera.add(rsMetadata.getColumnName(i));

}

lista.add(cabecera);

// para cada registro del resultset se añade una lista con sus

valores

// a la lista del resultset

int n = 1;

while (resultSet.next()) {

// si se supera el máximo de registros, se manda error

if (n++>ValoresPorDefecto.numMaxRecords)

throw new ResultSetException(

getText(

"Demasiados.registros.Filtrar"), resultSet);

List record = new ArrayList();

String urlParametro = "";

if (metodoLink) record.add(urlParametro);

for (int j = 1; j < i; j++) {

if (metodoLink) {

for (String linkParametro : linkParametros)

if (linkParametro.equals(

resultSet.getMetaData().getColumnName(j))) {

if (argumentos==null) {

if (resultSet.getObject(j)!=null)

urlParametro = urlParametro + "&"

+ linkParametro + "=" +

resultSet.getObject(j);

} else {

if (resultSet.getObject(j)!=null) {

urlParametro = urlParametro + "&"

+ argumentos + "=" +

linkParametro;

urlParametro = urlParametro + "&"

+ valores + "=" +

resultSet.getObject(j);

}

}

}

}

record.add(resultSet.getObject(j));

}

if (metodoLink) record.set(0,urlParametro.substring(1));

lista.add(record);

}

return lista;

}

Como paso final, solo queda presentar la información al usuario mediante un doble bucle que recorre la estructura List<List> incluyendo, si fuera necesario, elementos de link obtenidos a partir de la utilización de la anotación MetaDataLink en los métodos así requeridos de los metadata de la base de datos (en el trabajo solo se ha utilizado en el método getTables()). Tener en cuenta el tratamiento diferenciado de la primera fila como fila de títulos de la tabla html.

<s:iterator value="listInfo" var="record" status="status">

<s:if test="#status.first == true"><thead></s:if>

<tr>

<s:iterator value="#record" status="stat">

<s:set name="propiedad"><s:property /></s:set>

<s:if test="#status.first == true">

<s:if test="!metodoLink && #stat.count != 1">

<th><s:property /></th>

</s:if>

<s:elseif test="#stat.count != 1">

<th><s:property /></th>

</s:elseif>

<s:elseif test='#propiedad != null'>

<th class="ayuda"><s:property/></th>

</s:elseif>

</s:if>

<s:else>

<s:if test="metodoLink">

<s:if test="#stat.count == 1">

<s:url action="%{linkAction}"

namespace="%{linkNamespace}"

var="urlTag"/>

<s:set var="url">

<s:property value="urlTag"/>

?

<s:property/>

</s:set>

</s:if>

<s:elseif test="#stat.count == 2">

<td title='

<s:property

value="%{ayuda(propiedad)}"/>'>

<s:if test="#stat.count-1 ==

linkColumnNumber">

<s:a href="%{url}"

id="ir" >

<s:text

name="%{propiedad}"/>

</s:a>

</s:if>

<s:else>

<s:text

name="%{propiedad}"/>

</s:else>

</td>

</s:elseif>

<s:else>

<td>

<s:if test="#stat.count-1 ==

linkColumnNumber">

<s:a href="%{url}"

id="ir"><s:property/>

</s:a>

</s:if>

<s:else>

<s:property/>

</s:else>

</td>

</s:else>

</s:if>

<s:elseif test="#stat.count == 1">

<td title='<s:property

value="%{ayuda(#propiedad)}"/>'>

<s:text name="%{propiedad}"/>

</td>

</s:elseif>

<s:else>

<td>

<s:property/>

</td>

</s:else>

</s:else>

</s:iterator>

</tr>

<s:if test="#status.first == true"></thead></s:if>

</s:iterator>

También es interesante indicar como se construye el formulario html que permitirá al usuario enviar al sistema los valores de los argumentos de invocación de los métodos. El código se encuentra en filtro.jsp y, utilizando un iterador, permite adaptarse a cualquier número de argumentos, variable en función del método a invocar. Los campos se organizan en dos columnas:

<s:form name="filtro" method="POST" theme="simple" id="filtro">

<table class="filtro">

<s:iterator value="arrayParametros" status="stat">

<s:if test="#stat.odd == true">

<tr id="odd">

</s:if>

<s:set var="filtro">

<s:property escapeHtml="false"/>

</s:set>

<td>

<s:text name="%{filtro}" />:

<s:hidden name="filtroArgumentos" value="%{filtro}"/>

</td>

<td>

<s:textfield name="filtroValores"

value="%{getParameter(#stat.count-1)}"/>

</td>

<s:if test="#stat.even == true">

</tr>

</s:if>

</s:iterator>

</table>

<s:hidden name="metodo"/>

<s:hidden name="parametros"/>

<s:hidden name="TABLE\_SCHEM"/>

<s:hidden name="TABLE\_NAME"/>

<s:submit name="filtrar" value="filtrar"/>

</s:form>

El objeto arrayParametros referenciado es establecido en el método getInvoke.

### Trabajo con tablas

#### Consultar datos de una tabla

Esta sería la primera funcionalidad que se debería implementar, para presentar al usuario los registros filtrados de la tabla deseada. Los comandos de consulta en bases de datos relacionales utilizan el lenguaje SQL, y el comando particular de consulta es la sentencia SELECT. Esta, y todas las demás desarrolladas en este trabajo, están agrupadas en una clase diseñada a tal efecto denominada SQLCommand. Cualquier operación comenzará con la instanciación de un objeto de esta clase para lo cual es necesario pasar una conexión JDBC válida.

Después, solo deberemos invocar al método adecuado con los datos pertinentes para realizar la operación contra la base de datos oportuna.

Como se puede ver, el mapeado de la request lleva a la aplicación a ejecutar el método consulta() de la clase TablasAction, pero antes, se incluye en la validación el control de los datos necesarios para una correcta llamada posterior. En el método validate() se instancia un mapa con todos los campos de la tabla a consultar y se guarda en la sesión para no tener que realizarlo más veces para esa tabla:

public void validate() {

super.validarLogin(connectionImpl);

// si no hay errores, se carga el mapa de la tabla consultada

if (!hasActionErrors()) {

// busca el mapa del resultset en la sesión del usuario

mapaCompleto = (Map<String, Integer[]>)sesion.get(

TABLE\_SCHEM==null ? TABLE\_NAME :

(TABLE\_SCHEM.equals("") ? TABLE\_NAME :

TABLE\_SCHEM + "." + TABLE\_NAME));

// si el mapa aún no se ha creado

if (mapaCompleto == null) {

// se calcula y se guarda en la sesión

SQLCommand sqlCommand =

new SQLCommand(connectionImpl);

mapaCompleto = sqlCommand

.getMap(TABLE\_SCHEM, TABLE\_NAME);

sesion.put(

TABLE\_SCHEM==null ? TABLE\_NAME :

(TABLE\_SCHEM.equals("") ? TABLE\_NAME :

TABLE\_SCHEM + "." + TABLE\_NAME),

mapaCompleto);

}

}

}

A continuación, el método consulta() insta a la ejecución del comando SQL y al final carga una List<List> utilizando el método ya explicado getListInfo:

public String consulta() {

// Recoge los mensajes de error previos y los establece en

esta acción

if (sesion.get(ACTION\_ERROR)!=null) {

addActionError(getText((String)sesion.get(ACTION\_ERROR)));

sesion.remove(ACTION\_ERROR);

}

// Recoge los mensajes informativos previos y los establece en

esta acción

if (sesion.get(ACTION\_MESSAGE)!=null) {

addActionMessage(

getText((String)sesion.get(ACTION\_MESSAGE)));

sesion.remove(ACTION\_MESSAGE);

}

// comprueba que los parámetros se reciban correctamente

if (TABLE\_NAME == null || "".equals(TABLE\_NAME)) {

sesion.put(ACTION\_ERROR,

"Faltan.esquema.o.nombre.de.tabla");

return "back";

}

metodoLink = false; // Será true cuando exista una primary key

en la tabla consultada

// establece y recoge los parámetros de búsqueda en sesión del

usuario

Map<String, Integer> mapa = setParametrosSesion(

TABLE\_SCHEM==null ? TABLE\_NAME :

(TABLE\_SCHEM.equals("") ? TABLE\_NAME :

(TABLE\_SCHEM.equals("null") ? TABLE\_NAME :

TABLE\_SCHEM + "." + TABLE\_NAME)));

// define el array de posibles parámetros para presentarlos en

la jsp

arrayParametros = mapa.keySet();

//PreparedStatement ps = null;

ResultSet rs = null;

ResultSet rs2 = null;

try {

SQLCommand sqlCommand = new SQLCommand(connectionImpl);

rs = sqlCommand.executeQuery(TABLE\_SCHEM, TABLE\_NAME,

arrayParametros, sesion);

// se obtiene una lista con los campos primary key

List<String> pkList = new ArrayList<>();

String[] linkParametros = {};

rs2 = connectionImpl.getConnection().getMetaData()

.getPrimaryKeys(null,

TABLE\_SCHEM==null ? "" : TABLE\_SCHEM, TABLE\_NAME);

while (rs2.next())

pkList.add(rs2.getString("COLUMN\_NAME"));

if (pkList.size()!=0) metodoLink = true;

// se transforma el resultset en una lista para su

visualización

listInfo = getListInfo(rs, metodoLink,

pkList.toArray(linkParametros), "pkArgumentos",

"pkValores");

} catch (ResultSetException | SQLException |

SQLCommandException ex) {

addActionError(ex.getMessage());

return ERROR;

} finally {

cierraRS(rs, null);

cierraRS(rs2, null);

}

return SUCCESS;

}

En la pantalla tabla.jsp se presenta al usuario el conjunto de datos de una forma similar a la explicada en DatabaseMetaData.jsp. Solo comentar que, con el fin de enlazar la consulta con las operaciones de eliminación, edición o inserción, se incluyen a modo de ejemplo:

<ul class="bqY ocultar">

<s:url action="borrar" var="urlBru" includeParams="get"

escapeAmp="false"/>

<s:set var="url">

<s:property value="urlBru"/>&<s:property/>

</s:set>

<li class="bqX" title="<s:property

value='%{ayuda("borrar.registro")}'/>">

<s:a href="%{url}" class="acciones bru">&nbsp;&nbsp;</s:a>

</li>

En la clase SQLCommand, el método executeQuery nos permite obtener el resultset de la tabla:

public ResultSet executeQuery(String TABLE\_SCHEM,

String TABLE\_NAME, Set<String> arrayParametros,

Map<String, Object> sesion) throws SQLCommandException {

setSQLPartList(TABLE\_SCHEM, TABLE\_NAME, arrayParametros,

sesion);

return executeQuery(TABLE\_SCHEM, TABLE\_NAME,

ResultSet.TYPE\_FORWARD\_ONLY, ResultSet.CONCUR\_READ\_ONLY);

}

Que nos lleva al método setSQLPartList para informar las variables oportunas:

public void setSQLPartList(String TABLE\_SCHEM, String TABLE\_NAME,

Set<String> arrayParametros, Map<String, Object> sesion) {

inicializa();

// para cada columna del resultset

for (String columna : arrayParametros) {

// se busca en la sesión del usuario

Object o = sesion.get(getEsquema(TABLE\_SCHEM) + TABLE\_NAME

+ "." + columna);

// si el parámetro existe y no es "", se añade la

condición a la cadena where

// y el valor a la lista de parametros

if (o != null && !"".equals(o.toString())) {

// si no es el primer elemento se antepone AND

if (listaParametrosWhere.size() != 0) {

cadenaWhere = cadenaWhere + "AND ";

} // si es el primer filtro, se antepone where

else {

cadenaWhere = "where ";

}

// y a continuación se añade el nombre del campo y la

posición del parámetro.

// esto evitará ataques por SQL inyection

cadenaWhere = cadenaWhere + columna + " = ? ";

listaParametrosWhere.add(o);

}

}

}

Y al método executeQuery con los parámetros adecuados:

private ResultSet executeQuery(String TABLE\_SCHEM, String

TABLE\_NAME, int TYPE\_SCROLL, int CONCUR) throws

SQLCommandException {

// Se obtiene la cadena de consulta definitiva

String sql = String.format("SELECT \* FROM %s%s %s",

getEsquema(TABLE\_SCHEM), TABLE\_NAME, cadenaWhere);

PreparedStatement ps = null;

try {

// se obtiene el PreparedStatemtne

ps = connectionImpl.getConnection().prepareStatement(sql,

TYPE\_SCROLL, CONCUR);

// se asignan los parámetros

int parameterIndex = 1;

for (Object o : listaParametrosWhere) {

ps.setObject(parameterIndex++, o);

}

ResultSet rs = null;

// y se resuelve el resultset

return ps.executeQuery();

} catch (SQLException ex) {

throw new SQLCommandException(this, ex);

}

}

Finalmente, mostrar el formulario html que le permite al usuario aportar valores para realizar los filtros adecuados mediante el mismo código ya utilizado para la invocación de métodos del metadata de la base de datos.

#### Eliminar un registro de una tabla

public String borrar(){

if (TABLE\_NAME == null || "".equals(TABLE\_NAME)) {

sesion.put(ACTION\_ERROR,

"Faltan.esquema.o.nombre.de.tabla");

return "tablas";

}

if (pkArgumentos == null || pkArgumentos.length == 0) {

sesion.put(ACTION\_ERROR, "Falta.primary.key");

return "consulta";

}

try {

SQLCommand sqlComand = new SQLCommand(connectionImpl);

if (sqlComand.executeUpdate(TABLE\_SCHEM, TABLE\_NAME,

mapaCompleto, SQLCommand.OPERACION\_DELETE,

pkArgumentos, pkValores, null, null)==0) {

sesion.put(ACTION\_ERROR,

"No.se.ha.eliminado.ningún.registro");

} else {

sesion.put(ACTION\_MESSAGE,

"El.registro.ha.sido.eliminado");

}

} catch (SQLCommandException ex) {

sesion.put(ACTION\_ERROR, ex.getLocalizedMessage());

} finally {

return "consulta";

}

}

Con una llamada a SQLCommand.executeUpdate que se comentará en el punto siguiente

#### Editar un registro de una tabla

Con un paso a través de un formulario en el que se presentan los campos del registro, el usuario podrá en dicho formulario modificarlos y retornarlos al sistema para que se lance el correspondiente comando a la base de datos, teniendo en cuanta que necesitamos no solo los datos que han de ser modificados sino también los campos que conforman la primary key de la tabla.

public String editar(){

if (TABLE\_NAME == null || "".equals(TABLE\_NAME)) {

sesion.put(ACTION\_ERROR,

"Faltan.esquema.o.nombre.de.tabla");

return "tablas";

}

if (pkArgumentos == null || pkArgumentos.length == 0) {

sesion.put(ACTION\_ERROR, "Falta.primary.key");

return "consulta";

}

// obtiene el resultset del registro a editar

ResultSet rs = null;

try {

SQLCommand sqlComand = new SQLCommand(connectionImpl);

// y se resuelve el resultset

rs = sqlComand.executeQuery(TABLE\_SCHEM, TABLE\_NAME,

mapaCompleto, pkArgumentos, pkValores,

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY);

// se transforma el resultset en una lista para su

visualización

listInfo = getListInfoReverse(rs);

// se consigue el mapa de campos editables.

mapaEditables = sqlComand.getMap(mapaCompleto,

SQLCommand.ISDEFINITELYWRITABLE +

SQLCommand.ISWRITABLE);

} catch (SQLCommandException ex) {

addActionError(ex.getMessage());

return ERROR;

} finally {

cierraRS(rs,null);

}

return SUCCESS;

}

El proceso siguiente es el encargado de obtener los datos del formulario y guardarlos en la base de datos

public String editarGuardar() {

if (TABLE\_NAME == null || "".equals(TABLE\_NAME)) {

sesion.put(ACTION\_ERROR,

"Faltan.esquema.o.nombre.de.tabla");

return "tablas";

}

if (pkArgumentos == null || pkArgumentos.length == 0) {

sesion.put(ACTION\_ERROR, "Falta.primary.key");

return "consulta";

}

if (formCampos == null || formCampos.length == 0 ||

formValores == null || formValores.length == 0) {

sesion.put(ACTION\_ERROR,

"Faltan.datos.para.hacer.la.operacion");

return "consulta";

}

try {

SQLCommand sqlComand = new SQLCommand(connectionImpl);

if (sqlComand.executeUpdate(TABLE\_SCHEM, TABLE\_NAME,

mapaCompleto, SQLCommand.OPERACION\_UPDATE,

pkArgumentos, pkValores, formCampos, formValores)==0)

{

sesion.put(ACTION\_MESSAGE,

"No.se.ha.actualizado.ningún.registro");

} else {

sesion.put(ACTION\_MESSAGE,

"El.registro.ha.sido.actualizado");

}

} catch (SQLCommandException ex) {

sesion.put("ACTION\_ERROR", ex.getLocalizedMessage());

} finally {

return "consulta";

}

}

Que llama a la calse executeUpdate de SQLCommand:

public int executeUpdate(String esquema, String tabla,

Map<String, Integer[]> mapa, int operacion,

String[] pkArgumentos, String[] pkValores, String[] campos,

String[] valores) throws SQLCommandException {

int retorno = 0;

if (operacion == OPERACION\_UPDATE) {

setSQLPartList(mapa, pkArgumentos, pkValores, campos,

valores);

sql = String.format("UPDATE %s%s SET %s %s",

getEsquema(esquema),

tabla,

cadenaSet,

cadenaWhere);

} else if (operacion == OPERACION\_DELETE) {

setSQLPartList(mapa, pkArgumentos, pkValores,

OPERACION\_WHERE);

sql = String.format("DELETE FROM %s%s %s",

getEsquema(esquema),

tabla,

cadenaWhere);

}

if ("".equals(sql)) {

throw new SQLCommandException(this, operacion);

}

PreparedStatement ps = null;

try {

ps = connectionImpl.getConnection().prepareStatement(sql);

// se asignan los parámetros

int parameterIndex = 1;

// en primer lugar la parte update, no habrá parámetros si

no hay

// parte update

for (Object o : listaParametrosSet) {

ps.setObject(parameterIndex++, o);

}

// y luego la parte where

for (Object o : listaParametrosWhere) {

ps.setObject(parameterIndex++, o);

}

retorno = ps.executeUpdate();

} catch (SQLException ex) {

throw new SQLCommandException(this, ex);

} finally {

try {

if (ps != null) {

ps.close();

}

} catch (SQLException ex) {;

}

}

return retorno;

}

}

Y este a su vez hace dos llamadas al setSQLPartList:

private void setSQLPartList(Map<String, Integer[]> mapa,

String[] pkArgumentos, String[] pkValores, String[] campos,

String[] valores) {

inicializa();

int n = 0;

for (String columna : campos) {

Object o = getValor(mapa.get(columna)[0], valores[n++]);

if (listaParametrosSet.size() != 0) {

cadenaSet = cadenaSet + ", ";

}

cadenaSet = cadenaSet + columna + " = ? ";

listaParametrosSet.add(o);

}

n = 0;

for (String columna : pkArgumentos) {

Object o = getValor(mapa.get(columna)[0], pkValores[n++]);

if (listaParametrosWhere.size() != 0) {

cadenaWhere = cadenaWhere + "AND ";

} else {

cadenaWhere = "where ";

}

cadenaWhere = cadenaWhere + columna + " = ? ";

listaParametrosWhere.add(o);

}

}

#### Inserción de un registro en una tabla

De forma análoga se realizaría la funcionalidad de inserción de datos en una tabla de la base de datos.

## Compilación, instalación y ejecución del proyecto

[2]

### Conceptos básicos de ciclo de vida de construcción

Maven está basado en el concepto central del ciclo de vida de construcción de software lo que indica que el proceso para constuir y distribuir un proyecto está predefinido, es decir, que solo es necesario aprender un pequeño conjunto de comandos para construir cualquier proyecto de Maven y el fichero POM se asegurará de que se obtengan los resultados deseados.

Existen tres ciclos de vida de compilación incorporados: default, clean y site. El ciclo default maneja la implementación del proyecto, clean maneja la limpieza y site se encarga de la creación de la documentación del sitio del proyecto

#### Un ciclo de vida de construcción se compone de fases

Cada uno de estos ciclos está definido por una lista diferente de fases de compilación donde una fase representa una etapa en el ciclo de vida. Por ejemplo, el ciclo **default** se compone de las fases:

* **validate**: valida que el proyecto es correcto
* **compile**: Compila el código fuente del proyecto
* **test**: prueba el código fuente compilazo en un marco de prueba adecuado
* **package**: toma el código compilado y lo guarda en su formato distribuible
* **verify**: Verifica los resultados de las pruebas de integración
* **install**: instala el paquete en el repositorio local para usarlo como una dependencia en otros proyectos a nivel local
* **deploy**: copia el paquete final al repositorio remotor para compartirlo con otros desarrolladores

Estas fases se ejecutan de forma secuencia para completar el ciclo **default**.

#### Llamadas de línea de comandos habituales

En el entorno de desarrollo el siguiente comando crea e instala proyectos en el repositorio local:

mvn install

Este comando ejecutará cada fase predeterminada del ciclo de vida en orden (validate, compile, package, etc.) antes de ejecutar install.

El siguiente comando crea e implementa proyectos de forma limpia en el repositorio compartido:

mvn clean deploy

#### Una fase se compone de objetivos de complemento (plugin goals)

La manera en que se lleva a cabo las responsabilidades de un paso específico en el ciclo de vida de la compilación puede varia. Esto se hace declarando los objetivos del complemento vinculados a esas fases.

Un objetivo de complemento representa una tarea específica, más fina que una fase, que contribuye a la construcción y gestión de un proyecto. Puede estar vinculado a cero o más fases de construcción. El orden de ejecución de pende del orden en que se invocan los objetivos y las fases de creación. Por ejemplo, en el siguiente comando:

mvn clean dependency:copy-dependencies package

**clean** y **package** son fases, mientras **dependency:copy-dependencies** es un objetivo de complemento.

Si se ejecuta esto, la fase **clean** se ejecutará primero y luego el objetivo **dependency:copy-dependencies**, antes de finalmente ejecutar la fase **package**.

#### Algunas fases no se suelen llamar desde la línea de comandos

Las fases nombradas con palabras con guión (pre-\*, post-\* o process-\*) generalmente no se llaman directamente desde la línea de comando

### Configuración del proyecto

El ciclo de vida de la compilación es bastante sencillo de usar pero, ¿cómo asignamos tareas a cada una de las fases de compilación?

#### Packaging

Lo primero y más habitual es fijar el empaquetado del proyecto a través del elemento POM **<packaging>**. Algunos valores válidos para este término son **jar, war, ear** y **pom**. En nuestro caso, utilizaremos el valor **war**.

Cada paquete contiene una lista de objetivos para vincular a una fase particular.

#### Plugins

La segunda forma de agregar objetivos a las fases es configurar los complementos en su proyecto. Los plugins son proyectos que proporcionan ojetivos a Maven. Además, un plugin puede tener uno o más objetivos en los que cada objetivo representa una capacidad de ese plugin. Por ejemplo, el plugin de compilar contiene dos objetivos: compile y testCompile. El primero compila el código fuente y el segundo compila el código fuente de su código de prueba.

En nuestro proyecto es necesario indicar un plugin de compilación de maven que fuerce a realizar una compilación en versión java 1.8 y con un argumento de compilación específico: -parameters. Este argumento específico hace que en la fase de compilación, los argumentos de los métodos mantengan su nombre para que java Reflections pueda utilizarlos. Si no se hiciera, los argumentos tendrían los nombres genéricos arg0, arg1, etc.

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<compilerArguments>

<endorseddirs>${endorsed.dir}</endorseddirs>

</compilerArguments>

<compilerArgs>

<arg>-parameters</arg>

</compilerArgs>

</configuration>

</plugin>

### Referencia del ciclo de vida

A continuación se enumeran todas las fases de compilación de los ciclos de vida default, clean y site:

#### Ciclo de vida clean

| Fase | Descripción |
| --- | --- |
| pre-clean | Ejecutar los procesos necesarios antes de la limpieza real del proyecto. |
| clean | eliminar todos los archivos generados por la compilación anterior |
| post-clean | Ejecutar los procesos necesarios para finalizar el proyecto de limpieza. |

Tabla 39: ciclo de vida clean

#### Ciclo de vida default

| Fase | Descripción |
| --- | --- |
| validate | validar el proyecto es correcto y toda la información necesaria está disponible. |
| initialize | Inicializar el estado de compilación, por ejemplo, establecer propiedades o crear directorios. |
| generate-sources | Genera cualquier código fuente para incluirlo en la compilación. |
| process-sources | Procese el código fuente, por ejemplo para filtrar cualquier valor. |
| generate-resources | Generar recursos para su inclusión en el paquete. |
| process-resources | copie y procese los recursos en el directorio de destino, listo para empaquetar. |
| compile | compilar el código fuente del proyecto. |
| process-classes | postprocese los archivos generados a partir de la compilación, por ejemplo, para hacer una mejora del código de bytes en las clases de Java. |
| generate-test-sources | generar cualquier código fuente de prueba para su inclusión en la compilación. |
| process-test-sources | Procese el código fuente de prueba, por ejemplo, para filtrar cualquier valor. |
| generate-test-resources | Crear recursos para la prueba. |
| process-test-resources | Copie y procese los recursos en el directorio de destino de prueba. |
| test-compile | compile el código fuente de prueba en el directorio de destino de prueba |
| process-test-classes | postprocese los archivos generados a partir de la compilación de prueba, por ejemplo, para hacer una mejora del código de bytes en las clases de Java. Para Maven 2.0.5 y superior. |
| test | Ejecutar pruebas utilizando un marco de prueba de unidad adecuada. Estas pruebas no deben requerir que el código esté empaquetado o implementado. |
| prepare-package | realice las operaciones necesarias para preparar un paquete antes del embalaje real. Esto a menudo resulta en una versión desempaquetada y procesada del paquete. (Maven 2.1 y superior) |
| package | tomar el código compilado y empaquetarlo en su formato distribuible, como un JAR. |
| pre-integration-test | realizar las acciones necesarias antes de que se ejecuten las pruebas de integración. Esto puede implicar cosas como configurar el entorno requerido. |
| integration-test | procese y despliegue el paquete, si es necesario, en un entorno donde se puedan ejecutar pruebas de integración. |
| post-integration-test | realizar las acciones requeridas después de que las pruebas de integración hayan sido ejecutadas. Esto puede incluir la limpieza del medio ambiente. |
| verify | Ejecutar cualquier verificación para verificar que el paquete es válido y cumple con los criterios de calidad. |
| install | instale el paquete en el repositorio local, para usarlo como una dependencia en otros proyectos a nivel local. |
| deploy | hecho en un entorno de integración o lanzamiento, copia el paquete final al repositorio remoto para compartirlo con otros desarrolladores y proyectos. |

Tabla 40: ciclo de vida default

#### Ciclo de vida site

| Fase | Descripción |
| --- | --- |
| pre-site | Ejecute los procesos necesarios antes de la generación del sitio del proyecto real |
| site | Generar la documentación del sitio del proyecto. |
| post-site | Ejecutar los procesos necesarios para finalizar la generación del sitio y preparar la implementación del sitio. |
| site-deploy | implementar la documentación del sitio generado en el servidor web especificado |

Tabla 41: ciclo de vida site

### Encuadernaciones de ciclo de vida incorporadas

Algunas fases tienen objetivos vinculados a ellos por defecto. Y para el ciclo de vida default estos enlaces deenden del valor de empaquetado. Estos son algunos de los enlaces de objetivo a la fase de built-in

#### Enlaces de ciclo de vida clean

|  |  |
| --- | --- |
| Fase | plugin: objetivo |
| clean | clean: clean |

Tabla 42: Enlaces de ciclo de vida limpio

#### Enlaces de ciclo de vida default – packaging war

|  |  |
| --- | --- |
| Fase | plugin: objetivo |
| process-resources | resources: resources |
| compile | Compiler: compile |
| Process-test-resources | Resources: testResources |
| Test-compile | Compiler:testCompile |
| Test | Surefire:test |
| Package | War:war |
| Install | Install:install |
| Deploy | Deploy:deploy |

Tabla 43: Enlaces de ciclo de vida default - packaging war

#### Enlaces de ciclo de vida site

|  |  |
| --- | --- |
| Phase | plugin:goal |
| site | site:site |
| site-deploy | site:deploy |

Tabla 44: Enlaces de ciclo de vida site

#### Fijaciones de complementos para el packaging war

[3]

<phases>

<process-resources>

org.apache.maven.plugins:maven-resources-plugin:2.6:resources

</process-resources>

<compile>

org.apache.maven.plugins:maven-compiler-plugin:3.1:compile

</compile>

<process-test-resources>

org.apache.maven.plugins:maven-resources-plugin:2.6:testResources

</process-test-resources>

<test-compile>

org.apache.maven.plugins:maven-compiler-plugin:3.1:testCompile

</test-compile>

<test>

org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test

</test>

<package>

org.apache.maven.plugins:maven-war-plugin:2.2:war

</package>

<install>

org.apache.maven.plugins:maven-install-plugin:2.4:install

</install>

<deploy>

org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy

</deploy>

</phases>

## Pruebas del sistema

### Apache JMeter

[4]

La aplicación Apache JMeter es un software de código abierto, 100% java, diseñada para cargar el comportamiento funcional de la prueba y medir el rendimiento.

Se puede utilizar para probar el rendimiento tanto en recursos estáticos como dinámicos de aplicaciones web

Características:

* La capacidad de carga y el rendimiento prueban muchas aplicaciones, servidores y tipos de protocolo diferentes
* Permite la grabación rápida del plan de prueba
* Dispone de un modo de línea de comandos para cargar la prueba desde cualquier sistema operativo compatible Java
* Informe HTML completo
* Marco completo de subprocesos múltiples que permite el muestreo simultáneo por muchos subprocesos y el muestreo simultáneo de diferentes funciones por grupo de subprocesos separados.

No es un navegador, funciona a nivel de protocolo. Se parece a un navegador, sin embargo, JMeter no realiza todas las accciones admitidas por los navegadore, no ejecuta el javascript que se encuentra en las páginas HTML. Tampoco representa las páginas HTML como lo hace un navegador.

Se utilizó esta herramienta para realizar un paquete de pruebas y de carga de la aplicación suficiente, siempre teniendo en cuenta que el objetivo del proyecto es puramente educativo.

Se plantean cuatro request a la aplicación, una para establecer la conexión, otra ejecutando getTables, otra ejecutando getTables del esquema JHA y otra para hacer una select de una tabla. El resultado obtenido es:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
| HTTP Request conexion | 600 | 133 | 93 | 1936 | 84.85 | 0.00% | 2.69647 | 11.27 | 0.71 | 4278 |
| HTTP Request getTables | 600 | 90 | 59 | 800 | 48.34 | 0.00% | 2.71935 | 11.41 | 0.95 | 4297 |
| HTTP Request gettables jha | 600 | 48 | 34 | 296 | 22.79 | 100.00% | 2.72124 | 3.38 | 1.44 | 1273 |
| HTTP Request select contrato | 600 | 86 | 58 | 406 | 37.32 | 0.00% | 2.72168 | 11.42 | 0.55 | 4297 |
| TOTAL | 2400 | 89 | 34 | 1936 | 61.37 | 25.00% | 10.77722 | 37.22 | 3.61 | 3536.2 |

Tabla 45: Resultados pruebas JMeter

Apéndice E

# Documentación de usuario

## Introducción

Para que el usuario pueda utilizar la aplicación y comprenda el alcance y las posibilidades de la misma, se indica a continuación, en primer lugar, los requisitos de usuario que cumple, el sistema o procedimiento de instalación y un pequeño manual de usuario que le ayudará a utilizar la aplicación.

## Requisitos de usuario

Los requisitos de usuario vienen indicados en los objetivos con los que la aplicación se desarrolló. Brevemente los indicamos nuevamente:

* Permitir el mantenimiento de una base de datos remota vía internet
* Ofrecer un entorno amigable e intuitivo
* Permitir efectuar cualquier operación básica de mantenimiento sobre una base de datos relacional.
* Se dispondrá de una ayuda interactiva en línea

## Instalación

En el paquete de distribución se proporcionan dos posibilidades de instalación:

* Paquete war: Se proporciona un archivo empaquetado en formato war que se puede desplegar en una servidor web que cumpla los requisitos establecidos a tal efecto.
* Carpeta: Además se proporciona una carpeta con la aplicación compilada y la estructura de ficheros ya definida. Si se desea utilizar esta carpeta directamente, es necesario cargar un fichero xml que indique al servidor la ruta y el nombre que se le dará a la aplicación. A modo de ejemplo, podría ser:

<?xml version="1.0" encoding="UTF-8"?>

<Context docBase="ruta\target\myDataBaseJC-1.0-SNAPSHOT"

path="/myDataBaseJC"/>

## Manual de usuario

A continuación detallamos a modo de manual de usuario las distintas pantallas, normas, instrucciones y pasos que debe seguir un usuario para la correcta utilización de la aplicación

### Conexión a la base de datos

Al invocar por primera vez a la aplicación, o siempre que no se disponga de una conexión válida a la base de datos, el usuario visualizará la pantalla para introducir los datos de login

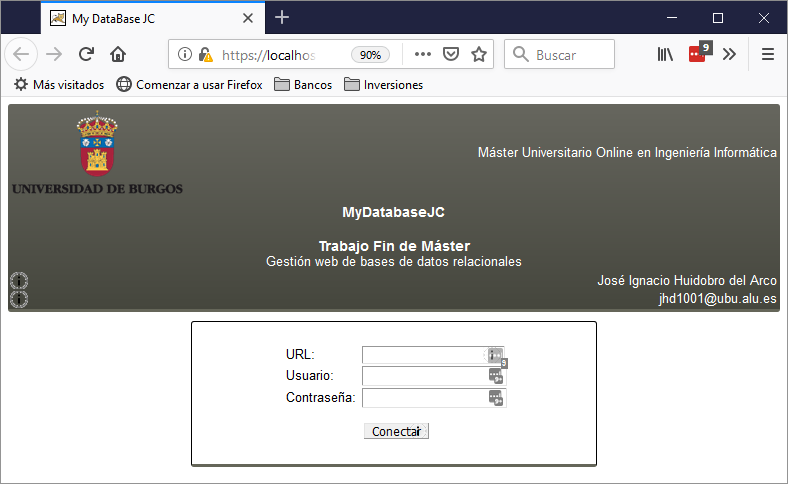


Figura 13: Pantalla de login

El usuario deberá introducir los datos solicitados tras lo que la aplicación intentará establecer la conexión. Si no puediera establecerse dicha conexión se informará al usuario de la causa en la misma pantalla.

Si el usuario obtiene la conexión a la base de datos se pasa al siguiente apartado

### Información inicial de la base de datos

Obtenida la conexión se le presenta al usuario la siguiente pantalla

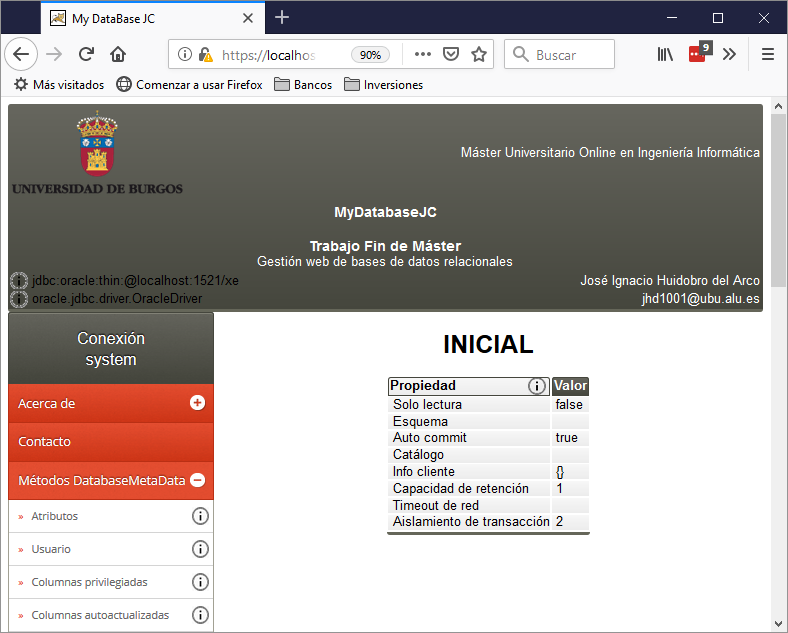


Figura 14: Pantalla inicial

En esta pantalla podemos distinguir tres zonas informativas:

* La banda superior muestra información general como universidad, nombre, aplicación, así como la url de conexión y el driver utilizado
* La banda lateral es un menú para que el usuario elija la operación a realizar. Además muestra en la parte superior el usuario de conexión
* La zona central muestra una colección de propiedades de la conexión establecida.

Cuando el usuario elije una determinada opción del menú lateral, en la zona central se presenta la información correspondiente.

El usuario puede colocar el ratón sobre aquellos elementos distinguidos con un símbolo de información (una i latina rodeada por un círculo) tras lo que aparecerá junto al cursor una explicación de lo referenciado a modo de ayuda interactiva o contextual.

Como caso especial, la opción de menú **Tablas**, muestra al usuario las tablas de la base de datos con unas determinados atributos:

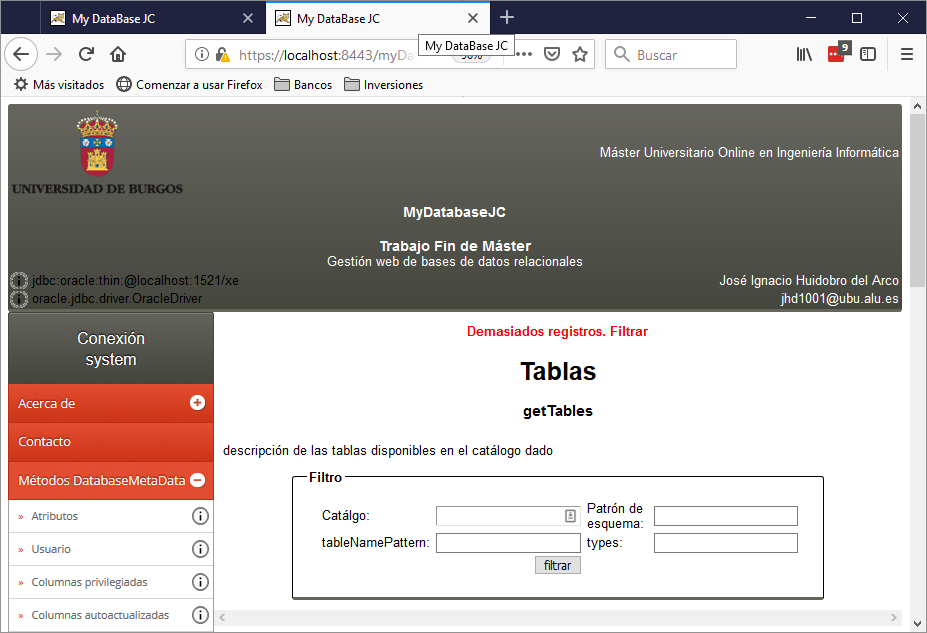


Figura 15: pantalla getTables sin argumentos

La primera vez que se utiliza esta opción, el mensaje indica que existen “demasiados registros. Filtrar”. Esto puede no ocurrir en todos los casos, depende del número de tablas que tenga la base de datos cuando no se filtra con ningún criterio. Si dispone de más de 1000 tablas, aparecerá este mensaje.

Se puede filtrar el número de tablas, en el caso que nos ocupa, o el número de registros en general de una opción de menú, ingresando los valores deseados en el formulario filtro que aparece. En nuestre ejemplo, si filtramos mediante el patrón de esquema JHA obtenemos la siguiente pantalla:

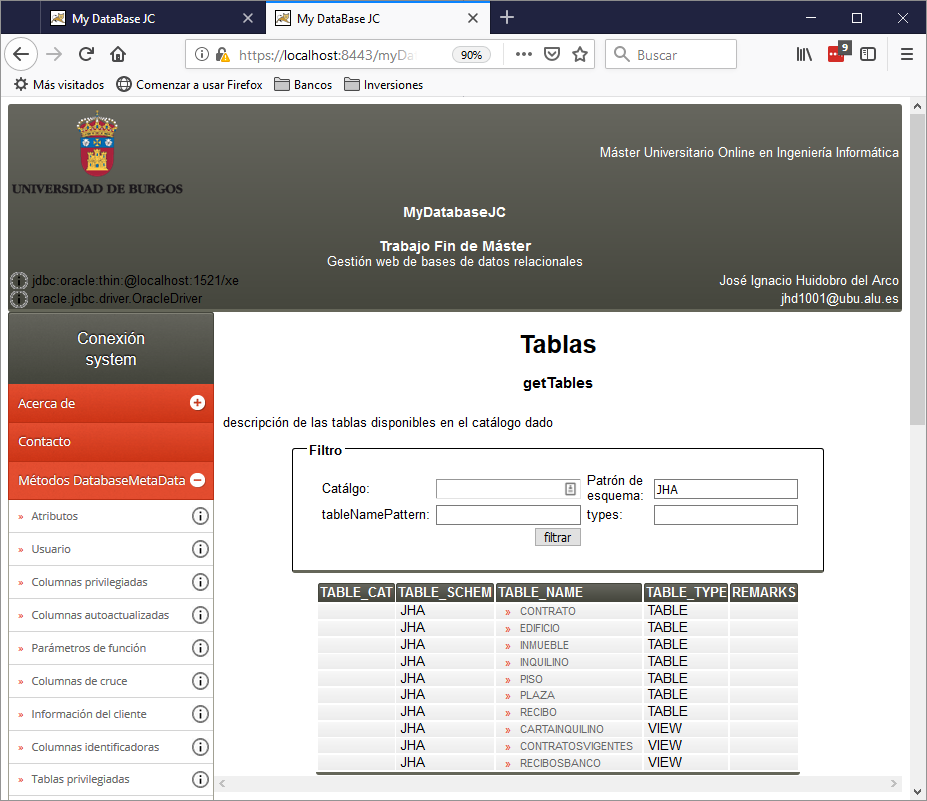


Figura 16: Pantalla getTables de un patrónn de esquema concreto

Como se puede observar, aparecen las tablas que cumplen con los criterios establecidos en el filtro.

Este sistema de filtrado, como ya se ha comentado, funciona con cualquiera de las funciones invocadas en el menú lateral.

Solo en el caso de Tablas, junto al TABLE\_NAME aparece un link que nos permitirá mostrar el contenido de una tabla concreta.

### Manteniemiento de una tabla

Si seguimos el link de alguna de las tablas visualizadas aparecerá la siguiente pantalla:

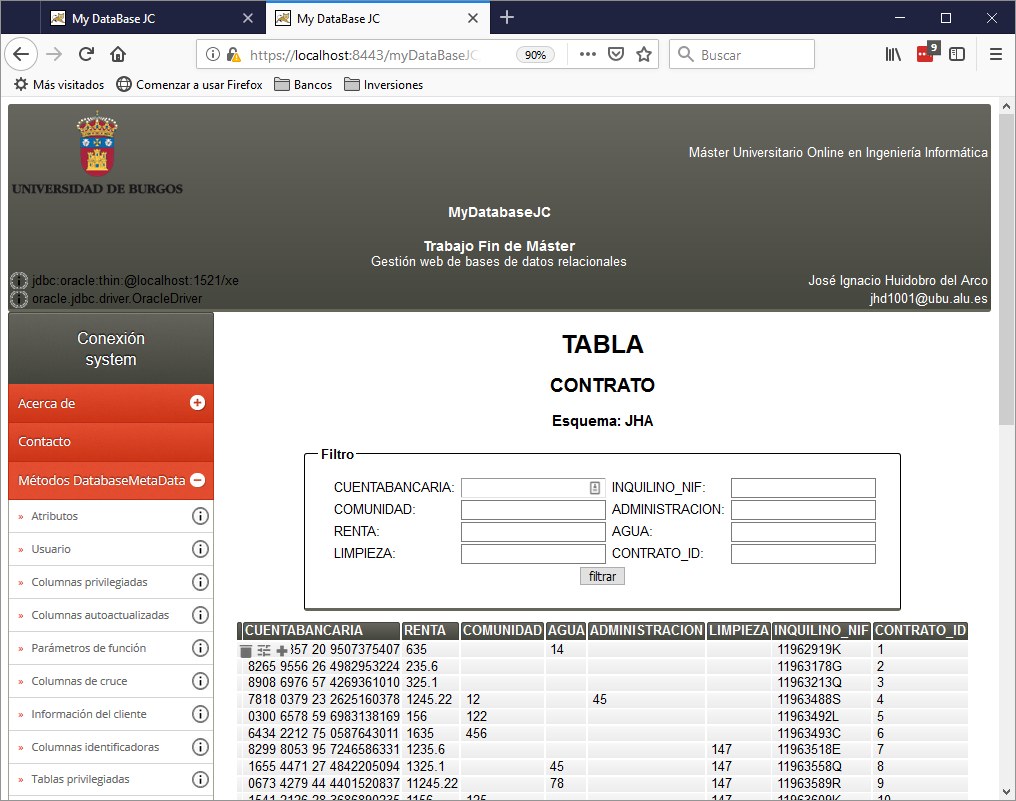


Figura 17: Pantalla de una consulta de tabla

De manera similar a lo indicado anteriormente, la zona central consta de un formulario para realizar filtrado de datos y una zona inferior en la que aparecen los datos de la tabla seleccionada.

En cada registro de la tabla, cuando recibe el foco del ratón, se activan 3 iconos[[1]](#footnote-1) que le permiten al usuario eliminar, editar o insertar un nuevo registro.

#### Modificar un registro

Al hacer clic en el icono de edición, aparece la siguiente pantalla:

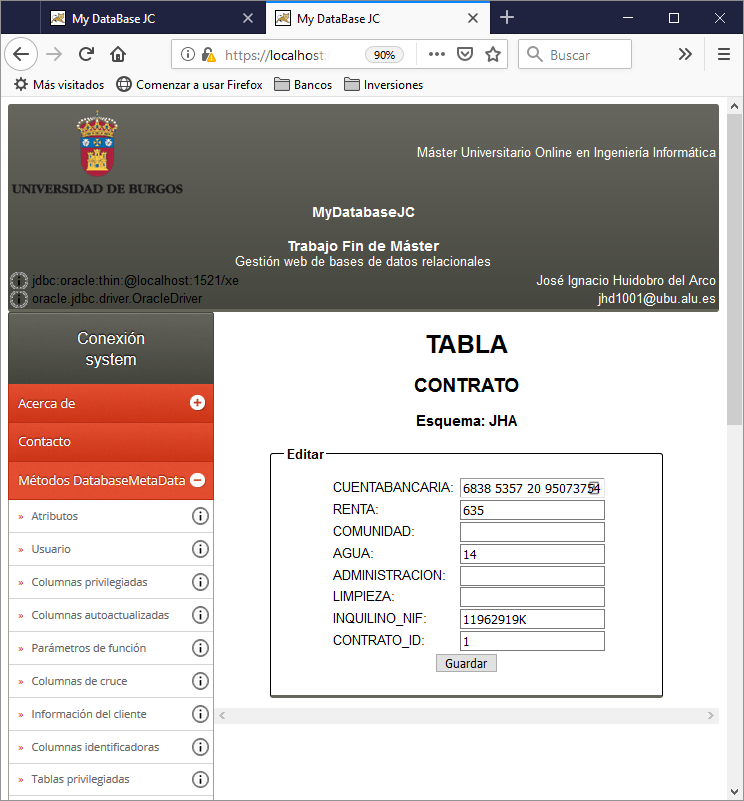


Figura 18: Editar registro de una tabla

En la pantalla se ve un formulario con todos los campos editables de la tabla, con los datos que actualmente están en el registro y la posibilidad de cambiarlos y finalmente, mediante el botón **Guardar**, grabar los cambios en la tabla correspondiente.

#### Insertar registro

Independientemente del registro en que se realice, un clic en el icono de insertar registro, aparecerá la siguiente pantalla:

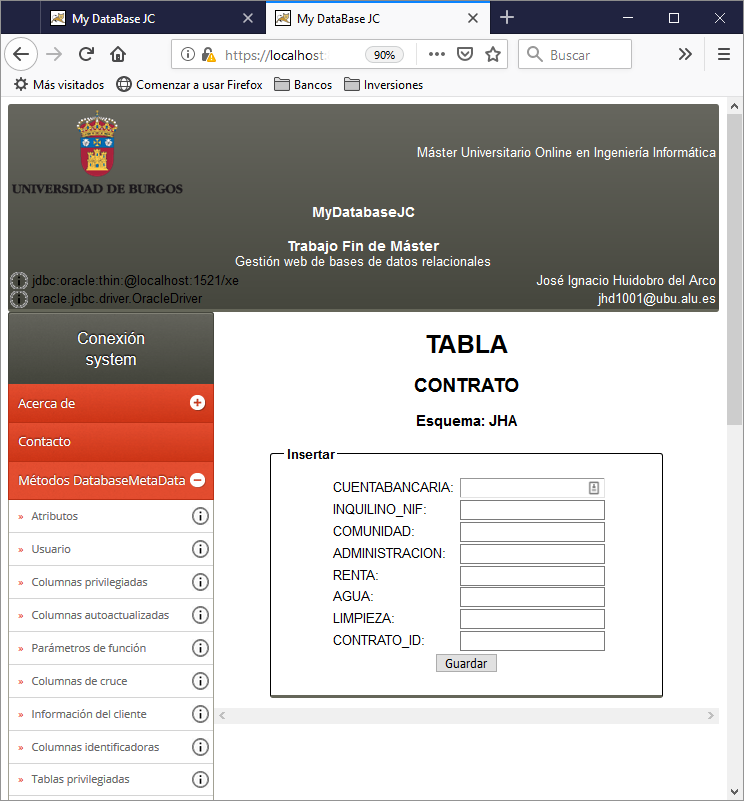


Figura 19: Insertar registro en una tabla

De manera similar a la pantalla de edició, el usuario completará los datos del registro y al hacer clic en **Guardar**, se almacenarán los datos en un nuevo registro en la tabla seleccionada.

Bibliografía

|  |  |  |
| --- | --- | --- |
| [1] | M. y. Barrado, «Especificación de Requisitos Software según el estándar IEEE 830,» 22 10 2008. [En línea]. Available: https://wikis.fdi.ucm.es/ELP/Especificaci%C3%B3n\_de\_Requisitos\_Software\_seg%C3%BAn\_el\_est%C3%A1ndar\_IEEE\_830. |  |
| [2] | The Apache Software Foundation, «Introduction to the Build Lifecycle,» 20 01 2019. [En línea]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html. |  |
| [3] | The Apache Software Foundation, «Plugin Bindings for default Lifecycle Reference,» 24 10 2018. [En línea]. Available: http://maven.apache.org/ref/3.6.0/maven-core/default-bindings.html. |  |
| [4] | The Apache Software Foundation, «Apache JMeter,» 2018. [En línea]. Available: https://jmeter.apache.org. |  |
| [6] | chuidiang, «DataBaseMetaData y ResultSetMetaData con java y MySQL,» 4 2 2007. [En línea]. Available: http://www.chuidiang.org/java/mysql/ResultSet-DataBase-MetaData.php. |  |
| [7] | Apache Software Fundatio, «Tomcat Web Application Deployment,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html. |  |
| [8] | Apache Software Foundation, «Security Considerations,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/security-howto.html. |  |
| [9] | Apache Software Foundation, «Apache Maven Compiler Plugin,» 26 7 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-compiler-plugin/. |  |
| [10] | Apache Software Foundation, «Apache Maven WAR Plugin,» 3 6 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-war-plugin/. |  |
| [11] | Apache Software Foundation, «Apache Maven Dependency Plugin,» 19 5 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-dependency-plugin/. |  |
| [12] | Apache Software Foundation, «Tomcat Setup,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/setup.html. |  |
| [13] | Apache Software Foundation, «Manager App HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html. |  |
| [14] | Apache Software Foundation, «JNDI Datasource HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html. |  |
| [15] | Apache Software Foundation, «Class Loader HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html. |  |
| [16] | Apache Software Foundation, «Connectors How To,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/connectors.html. |  |
| [17] | Programación en castellano, S.L., «Servlets básico,» [En línea]. Available: https://programacion.net/articulo/servlets\_basico\_108/2. |  |
| [18] | Stack Exchange, Inc., «What is the difference between javaee-api and javaee-web-api?,» 2013. [En línea]. Available: https://stackoverflow.com/questions/16789020/what-is-the-difference-between-javaee-api-and-javaee-web-api. |  |
| [19] | Fundación Wikimedia, Inc., «Java EE,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Java\_EE. |  |
| [20] | Fundación Wikimedia, Inc., «Java Servlet,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Java\_Servlet. |  |
| [21] | Fundación Wikimedia, Inc., «Java Server Page,» 12 06 2018. [En línea]. Available: https://es.wikipedia.org/wiki/JavaServer\_Pages. |  |
| [22] | Oracle Corporation, «Java JDBC API,» 2018. [En línea]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/. |  |
| [23] | Oracle Corporation, «Industry Support,» [En línea]. Available: https://www.oracle.com/technetwork/java/index-136695.html. |  |
| [24] | Oracle Corporation, «JDBC Overview,» [En línea]. Available: https://www.oracle.com/technetwork/java/overview-141217.html. |  |
| [25] | Oracle Corporation, «Features,» [En línea]. Available: https://www.oracle.com/technetwork/java/features-135464.html. |  |
| [26] | L. Hernández, «API JDBC como interfaz de acceso a bases de datos SQL,» 28 9 2004. [En línea]. Available: http://www.iuma.ulpgc.es/users/lhdez/inves/pfcs/memoria-ivan/node8.html. |  |
| [27] | Fundación Wikipedia, inc., «Tomcat,» 15 12 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Tomcat. |  |
| [28] | Wikimedia Foundation, Inc., «Apache Struts 2,» 18 11 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Apache\_Struts\_2. |  |
| [29] | Apache Software Foundation, «Key Technologies Primer,» 2018. [En línea]. Available: https://struts.apache.org/primer.html. |  |
| [30] | Apache Software Foundation, «How To Create A Struts 2 Web Application,» 2018. [En línea]. Available: https://struts.apache.org/getting-started/how-to-create-a-struts2-web-application.html. |  |
| [31] | Wkimedia Foundation, Inc., «Maven,» 8 1 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Maven. |  |
| [32] | Wikimedia Foundation, Inc., «Netbeans,» 28 12 2018. [En línea]. Available: https://es.wikipedia.org/wiki/NetBeans. |  |
| [33] | Apache Software Foundation, «Creating an Enterprise Application Using Maven,» 2018. [En línea]. Available: https://netbeans.org/kb/docs/javaee/maven-entapp.html. |  |

1. Sólo si la tabla dispone de una primary key [↑](#footnote-ref-1)